Fall 2019
EE 6633: Architecture of Parallel Computers
Lecture 6: Dataflow and Systolic Architectures

Avinash Karanth
Department of Electrical Engineering & Computer Science
Ohio University, Athens, Ohio 45701
E-mail: karanth@ohio.edu
Website: http://ace.cs.ohio.edu/~avinashk/classes/ee663/ee663.htm

1

Topics

- Dataflow Architectures
- Systolic Architectures

2

Data Flow

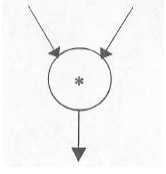
- The models we have examined in 447/740 all assumed
  - Instructions are fetched and retired in sequential, control flow order
- This is part of the Von-Neumann model of computation
  - Single program counter
  - Sequential execution
  - Control flow determines fetch, execution, commit order
- What about out-of-order execution?
  - Architecture level: Obeys the control-flow model
  - Uarch level: A window of instructions executed in data-flow order → execute an instruction when its operands become available

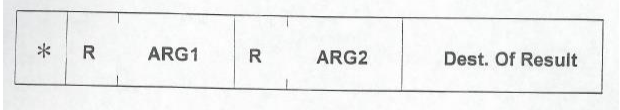
3

3

Data Flow

- In a data flow machine, a program consists of data flow nodes
- A data flow node fires (fetched and executed) when all its inputs are ready
  - i.e. when all inputs have tokens
- Data flow node and its ISA representation

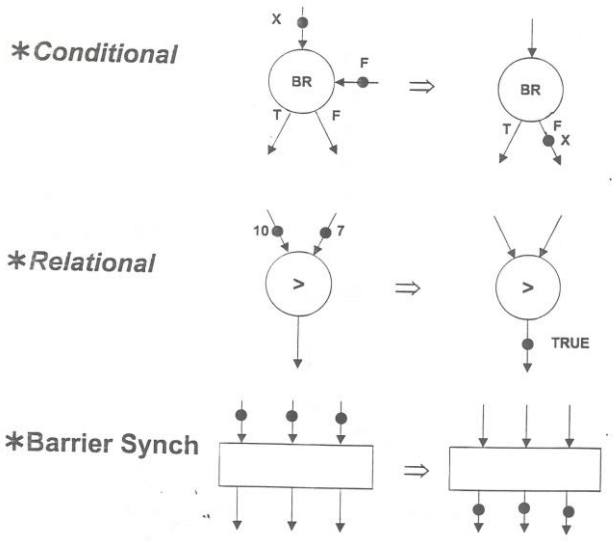




4

4

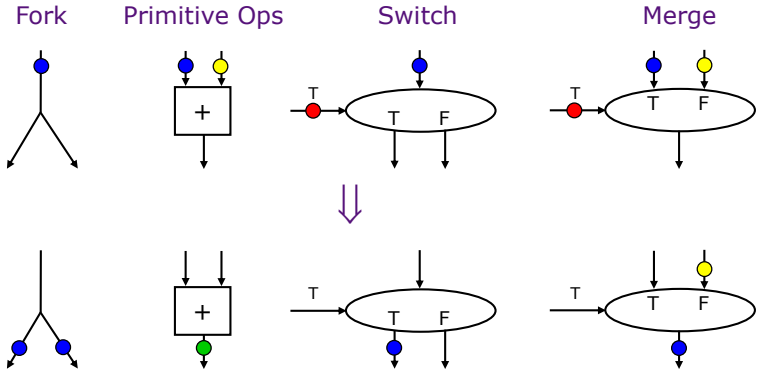
Data Flow Nodes



5

Data Flow Nodes (II)

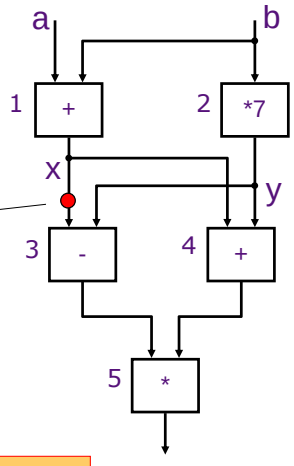
- A small set of dataflow operators can be used to define a general programming language



6

Dataflow Graphs

```
{x = a + b;
 y = b * 7
in
   (x-y) * (x+y)}
```



- Values in dataflow graphs are represented as tokens

token < ip , p , v >

ip = 3
p = L

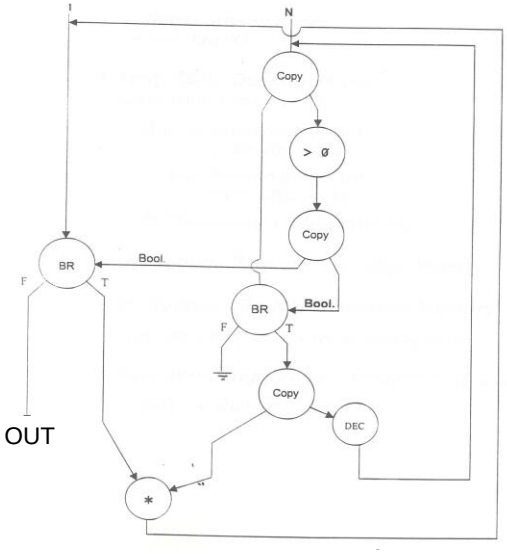
instruction ptr port data

- An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators

no separate control flow

7

Example Data Flow Program



8

Control Flow vs. Data Flow

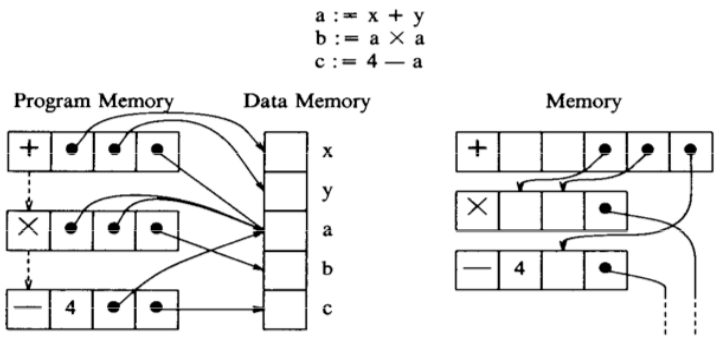


Figure 2. A comparison of control flow and dataflow programs. On the left a control flow program for a computer with memory-to-memory instructions. The arcs point to the locations of data that are to be used or created. Control flow arcs are indicated with dashed arrows; usually most of them are implicit. In the equivalent dataflow program on the right only one memory is involved. Each instruction contains pointers to all instructions that consume its results.

9

9

Class Question

- Implement the following function in a dataflow graph:

  $\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6!$

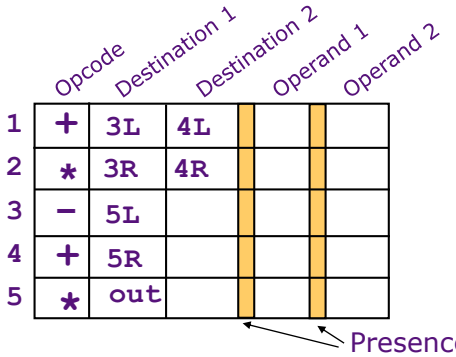
10

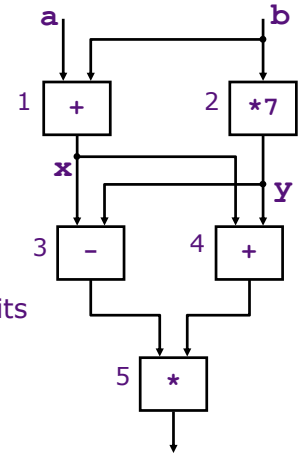
Static Dataflow

- Allows only one instance of a node to be enabled for firing
- A dataflow node is fired only when all of the tokens are available on its input arcs and no tokens exist on any of its its output arcs
- Dennis and Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," ISCA 1974.

11

Static Dataflow Machine:

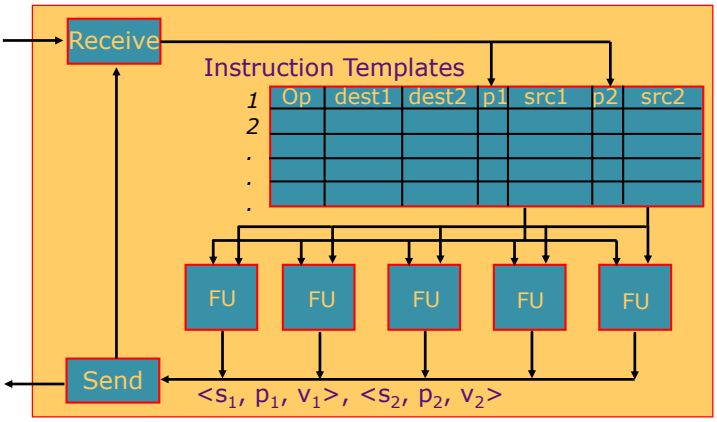




Each arc in the graph has an operand slot in the program

12

Static Dataflow Machine (Dennis+, ISCA 1974)



- Many such processors can be connected together
- Programs can be statically divided among the processors

13

Static Data Flow Machines

- Mismatch between the model and the implementation
  - The model requires *unbounded FIFO token queues* per arc but the architecture provides storage for one token per arc
  - The architecture *does not ensure FIFO* order in the reuse of an operand slot
- The static model *does not support*
  - Reentrant code
    - Function calls
    - Loops
  - Data Structures

14

14

Problems with Re-entrancy

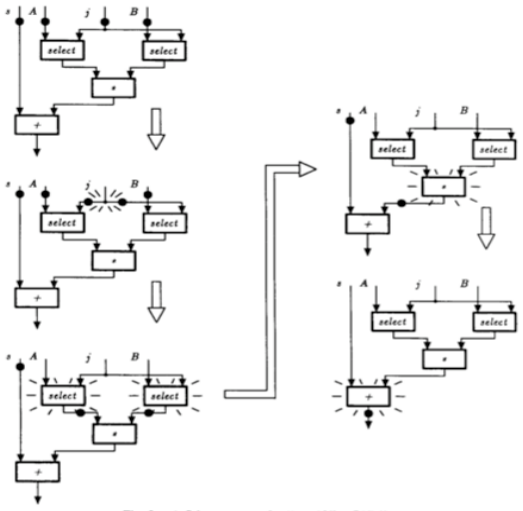


Fig. 3. A firing sequence for "$s+A[i]*B[i]$."

- Assume this was in a loop
- Or in a function
- And operations took variable time to execute
- How do you ensure the tokens that match are of the same invocation?

15

Static versus Dynamic Dataflow Machines

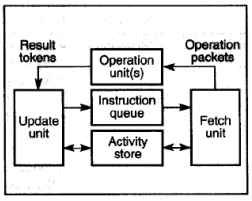


Figure 1. The basic organization of the static dataflow model.

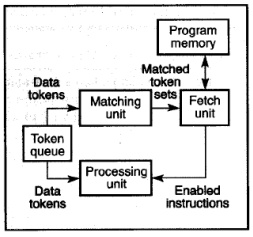


Figure 3. The general organization of the dynamic dataflow model.

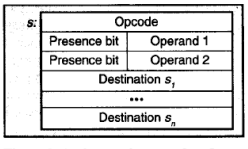


Figure 2. An instruction template for the static dataflow model.

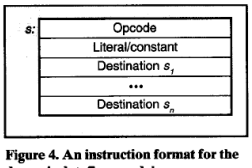


Figure 4. An instruction format for the dynamic dataflow model.

16

16

Dynamic Dataflow Architectures

- Allocate instruction templates, i.e., a frame, dynamically to support each loop iteration and procedure call
  - termination detection needed to deallocate frames
- The code can be shared if we separate the code and the operand storage

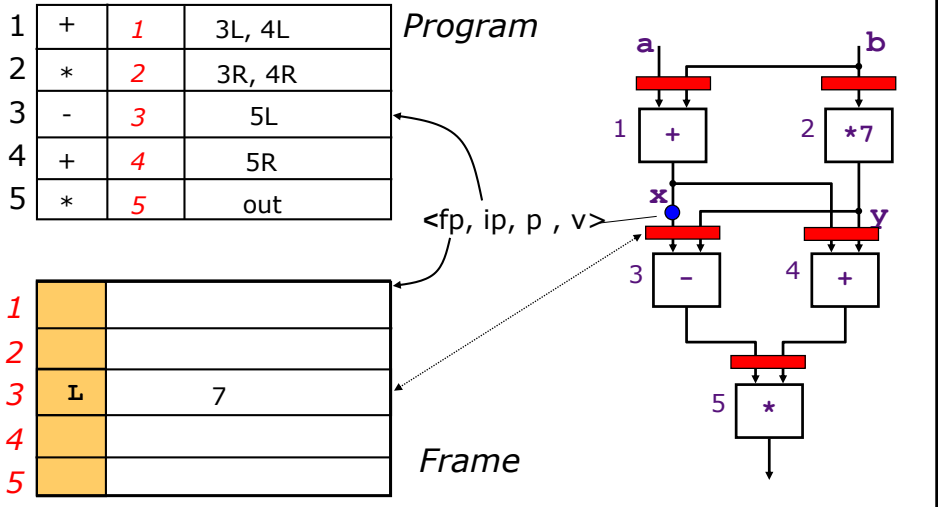
a token <fp, ip, port, data>

frame pointer

instruction pointer

17

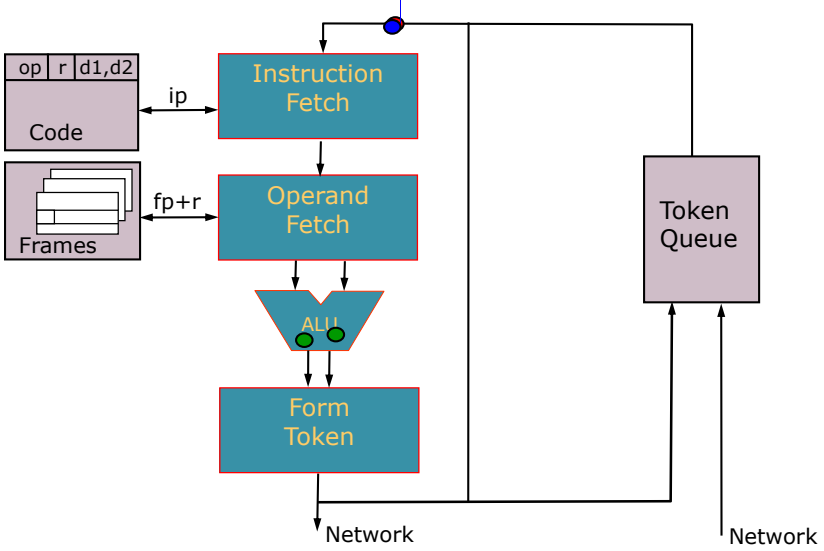
A Frame in Dynamic Dataflow



Need to provide storage for only one operand/operator

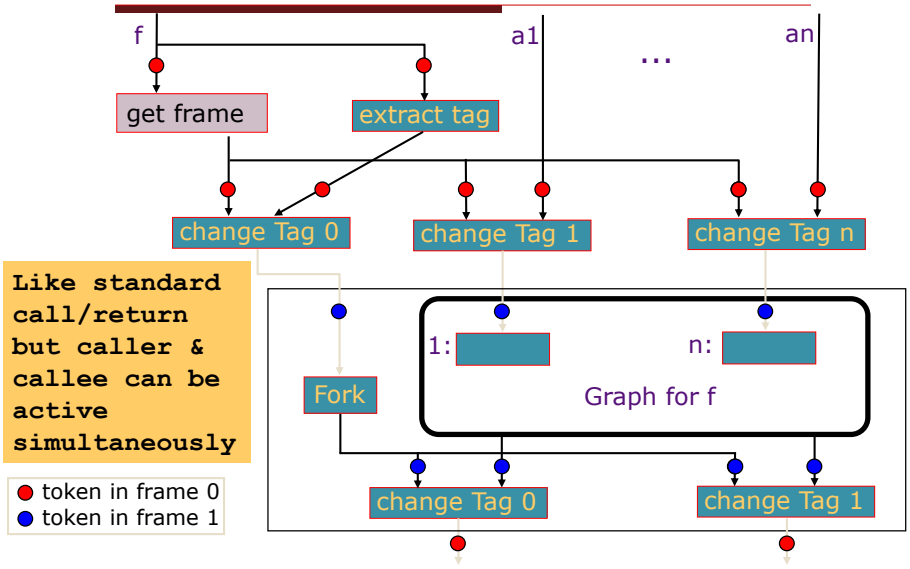
18

Monsoon Processor (ISCA 1990)



19

Procedure Linkage Operators



20

Control of Parallelism

- Problem: Many loop iterations can be present in the machine at any given time
  - 100K iterations on a 256 processor machine can swamp the machine (thrashing in token matching units)
  - Not enough bits to represent frame id
- Solution: Throttle loops. Control how many loop iterations can be in the machine at the same time.
  - Requires changes to loop dataflow graph to inhibit token generation when number of iterations is greater than N

21

21

Data Structures

- Dataflow by nature has write-once semantics
- Each arc (token) represents a data value
- An arc (token) gets transformed by a dataflow node into a new arc (token) → No persistent state…
- Data structures as we know of them (in imperative languages) are structures with persistent state
- Why do we want persistent state?
  - More natural representation for some tasks? (bank accounts, databases, …)
  - To exploit locality

22

22

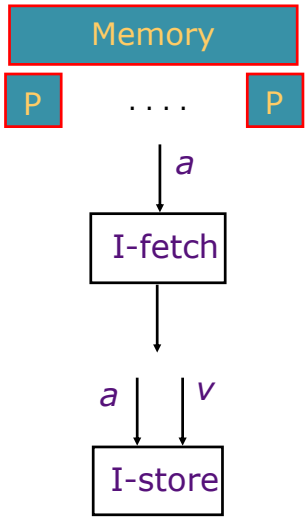
Data Structures in Dataflow

- Data structures reside in a structure store

  ⇒ tokens carry pointers

- I-structures: Write-once, Read multiple times *or*
  - allocate, write, read, ..., read, deallocate

  ⇒ No problem if a reader arrives before the writer at the memory location



23

I-Structures

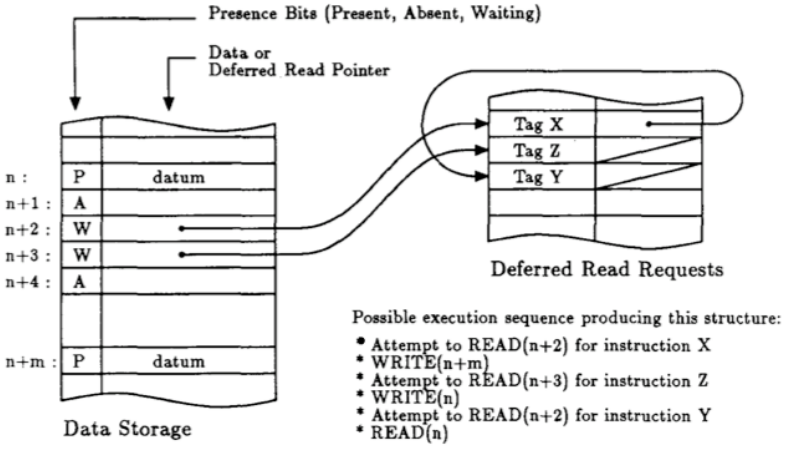
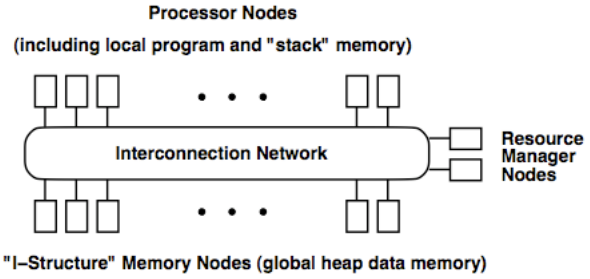


Fig. 7. *I*-structure memory.

24

24

MIT Tagged Token Data Flow Architecture

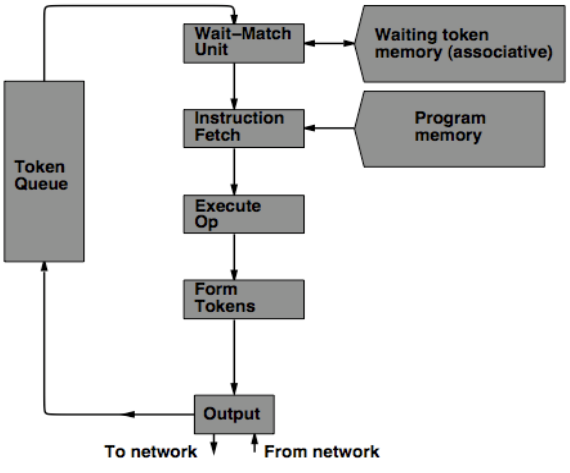


- Resource Manager Nodes
  - responsible for Function allocation (allocation of context/frame identifiers), Heap allocation, etc.

25

25

MIT Tagged Token Data Flow Architecture



- Wait–Match Unit: try to match incoming token and context id and a waiting token with same instruction address
  - Success: Both tokens forwarded
  - Fail: Incoming token –––> Waiting Token Mem, bubble (no-op forwarded)

26

26

Data Flow Summary

- Availability of data determines order of execution
- A data flow node fires when its sources are ready
- Programs represented as data flow graphs (of nodes)
- Data Flow at the ISA level has not been (as) successful
- Data Flow implementations under the hood (while preserving sequential ISA semantics) have been successful
  - Out of order execution
  - Hwu and Patt, "HPSm, a high performance restricted data flow architecture having minimal functionality," ISCA 1986.

Data Flow Characteristics

- Data-driven execution of instruction-level graphical code
  - Nodes are operators
  - Arcs are data (I/O)
  - As opposed to control-driven execution
- Only real dependencies constrain processing
- No sequential I-stream
  - No program counter
- Operations execute asynchronously
- Execution triggered by the presence of data
- Single assignment languages and functional programming
  - E.g., SISAL in Manchester Data Flow Computer
  - No mutable state

Data Flow Advantages/Disadvantages

- Advantages
  - Very good at exploiting irregular parallelism
  - Only real dependencies constrain processing
- Disadvantages
  - Debugging difficult (no precise state)
    - Interrupt/exception handling is difficult (what is precise state semantics?)
  - Implementing dynamic data structures difficult in pure data flow models
  - Too much parallelism? (Parallelism control needed)
  - High bookkeeping overhead (tag matching, data storage)
  - Instruction cycle is inefficient (delay between dependent instructions), memory locality is not exploited

Topics

- Dataflow Architectures
- Systolic Architectures

Why Systolic Architectures?

- Idea: Data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory
- Similar to an assembly line
  - Different people work on the same car
  - Many cars are assembled simultaneously
  - Can be two-dimensional
- Why? Special purpose accelerators/architectures need
  - Simple, regular designs (keep # unique parts small and regular)
  - High concurrency → high performance
  - Balanced computation and I/O (memory access)

31

31

Systolic Architectures

- H. T. Kung, "Why Systolic Architectures?," IEEE Computer 1982.

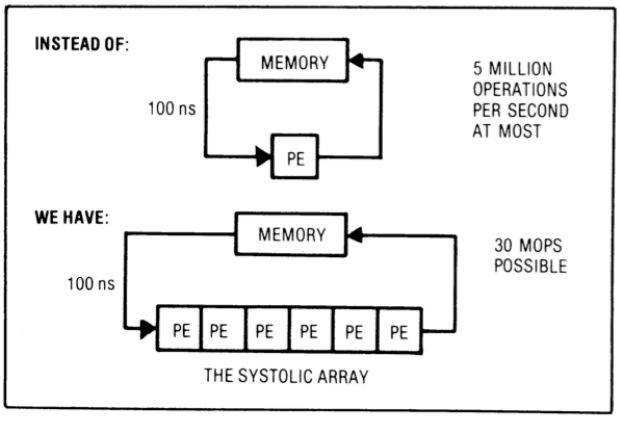


Figure 1. Basic principle of a systolic system.

Memory: heart
PEs: cells

Memory pulses data through cells

32

32

Systolic Architectures

- Basic principle: Replace a single PE with a regular array of PEs and carefully orchestrate flow of data between the PEs → achieve high throughput w/o increasing memory bandwidth requirements

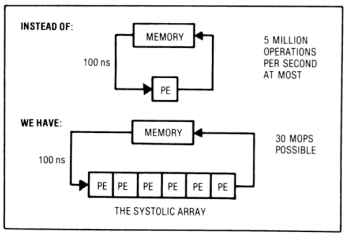


Figure 1. Basic principle of a systolic system.

- Differences from pipelining:
  - Array structure can be non-linear and multi-dimensional
  - PE connections can be multidirectional (and different speed)
  - PEs can have local memory and execute kernels (rather than a piece of the instruction)

33

Systolic Computation Example

- Convolution
  - Used in filtering, pattern matching, correlation, polynomial evaluation, etc …
  - Many image processing tasks

Given the sequence of weights $\{w_1, w_2, \ldots, w_k\}$
and the input sequence $\{x_1, x_2, \ldots, x_n\}$,

compute the result sequence $\{y_1, y_2, \ldots, y_{n+1-k}\}$
defined by

$$y_i = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1}$$

34

34

Systolic Computation Example: Convolution

- y1 = w1x1 + w2x2 + w3x3
- y2 = w1x2 + w2x3 + w3x4
- y3 = w1x3 + w2x4 + w3x5

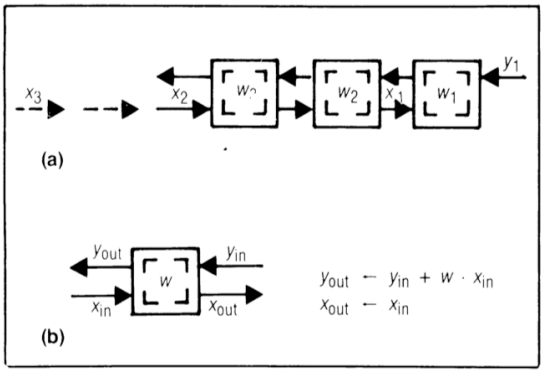


Figure 8. Design W1: systolic convolution array (a) and cell (b) where w_i's stay and x_i's and y_i's move systolically in opposite directions.

35

35

Systolic Computation Example: Convolution

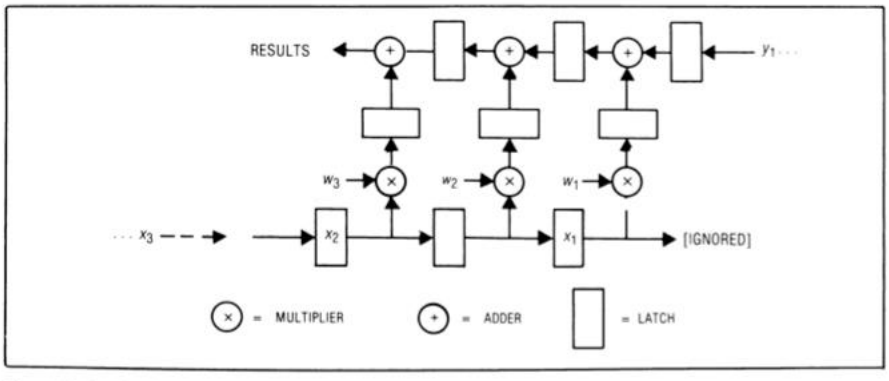


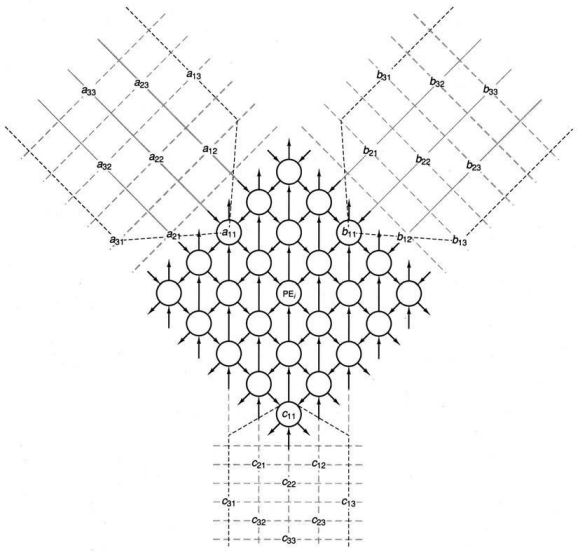
Figure 10. Overlapping the executions of multiply and add in design W1.

- Worthwhile to implement adder and multiplier separately to allow overlapping of add/mul executions

36

36

Hexagonal Array



$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

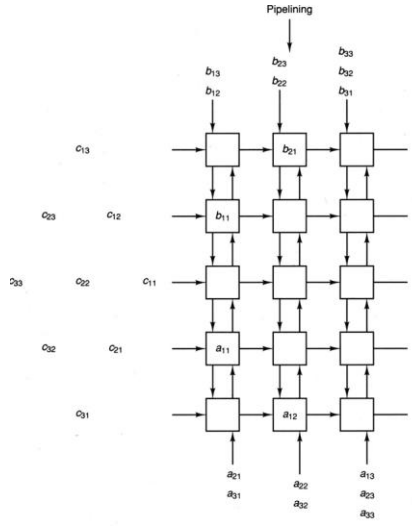
$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$c_{11} = c_{11} + a_{11} * b_{11}.$$

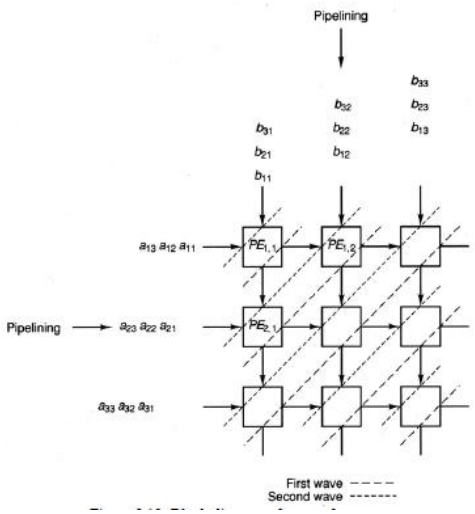
37

Pipelined Array



38

Wavefront Array



39

More Programmability

- Each PE in a systolic array
  - Can store multiple "weights"
  - Weights can be selected on the fly
  - Eases implementation of, e.g., adaptive filtering
- Taken further
  - Each PE can have its own data and instruction memory
  - Data memory → to store partial/temporary results, constants
  - Leads to stream processing, pipeline parallelism
    - More generally, staged execution

40

40

Pipeline Parallelism

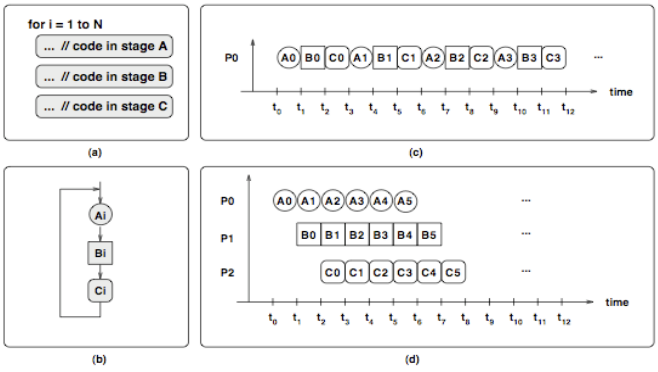


Figure 1. (a) The code of a loop, (b) Each iteration is split into 3 pipeline stages: A, B, and C. Iteration i comprises Ai, Bi, Ci. (c) Sequential execution of 4 iterations. (d) Parallel execution of 6 iterations using pipeline parallelism on a three-core machine. Each stage executes on one core.

41

41

File Compression Example

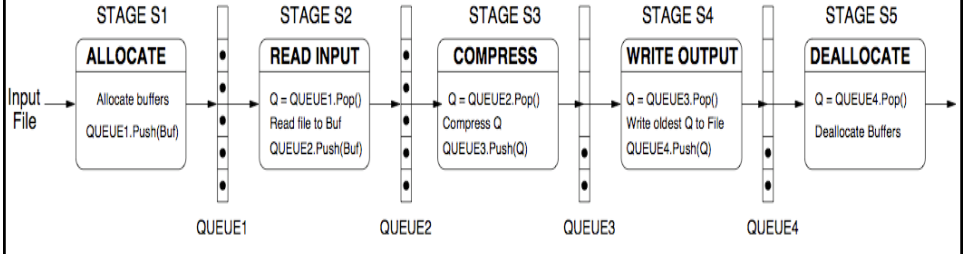


Figure 3. File compression algorithm executed using pipeline parallelism

42

42

Systolic Array

- Advantages
  - Makes multiple uses of each data item → reduced need for fetching/refetching
  - High concurrency
  - Regular design (both data and control flow)
- Disadvantages
  - Not good at exploiting irregular parallelism
  - Relatively special purpose → need software, programmer support to be a general purpose model