Problems of Asynchrony

Submitted by
Maria Baby
S7,CSE
Roll No: 42

Asynchronous

- In computer programs, asynchronous operation means that a process operates independently of other processes, whereas synchronous operation means that the process runs only as a result of some other process being completed or handing off operation.
- A typical activity that might use a synchronous protocol would be a transmission of files from one point to another.

- As each transmission is received, a response is returned indicating success or the need to resend. Each successive transmission of data requires a response to the previous transmission before a new one can be initiated.
- In telecommunication signaling within a network or between networks, an asynchronous signal is one that is transmitted at a different clock rate than another signal.

- Asynchronous data transfer: sender provides a synchronization signal to the receiver before starting the transfer of each message
- does not need clock signal between the sender and the receiver
- slower data transfer rate

Advantages

- The character is self contained & Transmitter and receiver need not be synchronized
- Transmitting and receiving clocks are independent of each other

Disadvanatges

- Overhead of start and stop bits
- False recognition of these bits due to noise on the channel

Applications

- If channel is reliable, then suitable for high speed else low speed transmission
- Most common use is in the ASCII terminals

Multithreading Solutions and Distributed Cacheing

Submitted by
Marvel Monson
S7,CSE
Roll No : 43

Multithreading

- Multithreading is a type of execution model that allows multiple threads to exist within the context of a process such that they execute independently but share their process resources.
- A thread maintains a list of information relevant to its execution including the priority schedule, exception handlers, a set of CPU registers, and stack state in the address space of its hosting process.
- Multithreading is also known as threading.

Operating systems use threading in two ways:

- Pre-emptive multithreading, in which the context switch is controlled by the operating system. Context switching might be performed at an inappropriate time, Hence, a high priority thread could be indirectly pre-empted by a low priority thread.
- Cooperative multithreading, in which context switching is controlled by the thread. This could lead to problems, such as deadlocks, if a thread is blocked waiting for a resource to become free.

Distributed Caching

- Caching has become the de facto technology to boost application performance as well as reduce costs.
- By caching frequently used data in memory – rather than making database round trips – application response times can be dramatically improved.

- Distributed caching is simply an extension of this concept, but the cache is configured to span multiple servers.
- It's commonly used in cloud computing and virtualised environments, where different servers give a portion of their cache memory into a pool which can then be accessed by virtual machines. This also means it’s a much more scalable option.

What makes distributed caching effective?

- Performance
- Scalability
- Availability
- Manageability
- Simplicity
- Affordability

CONTEXT SWITCHING IN MULTITHREADING

By,
MERIN JOSEPH
S7 CSE
45

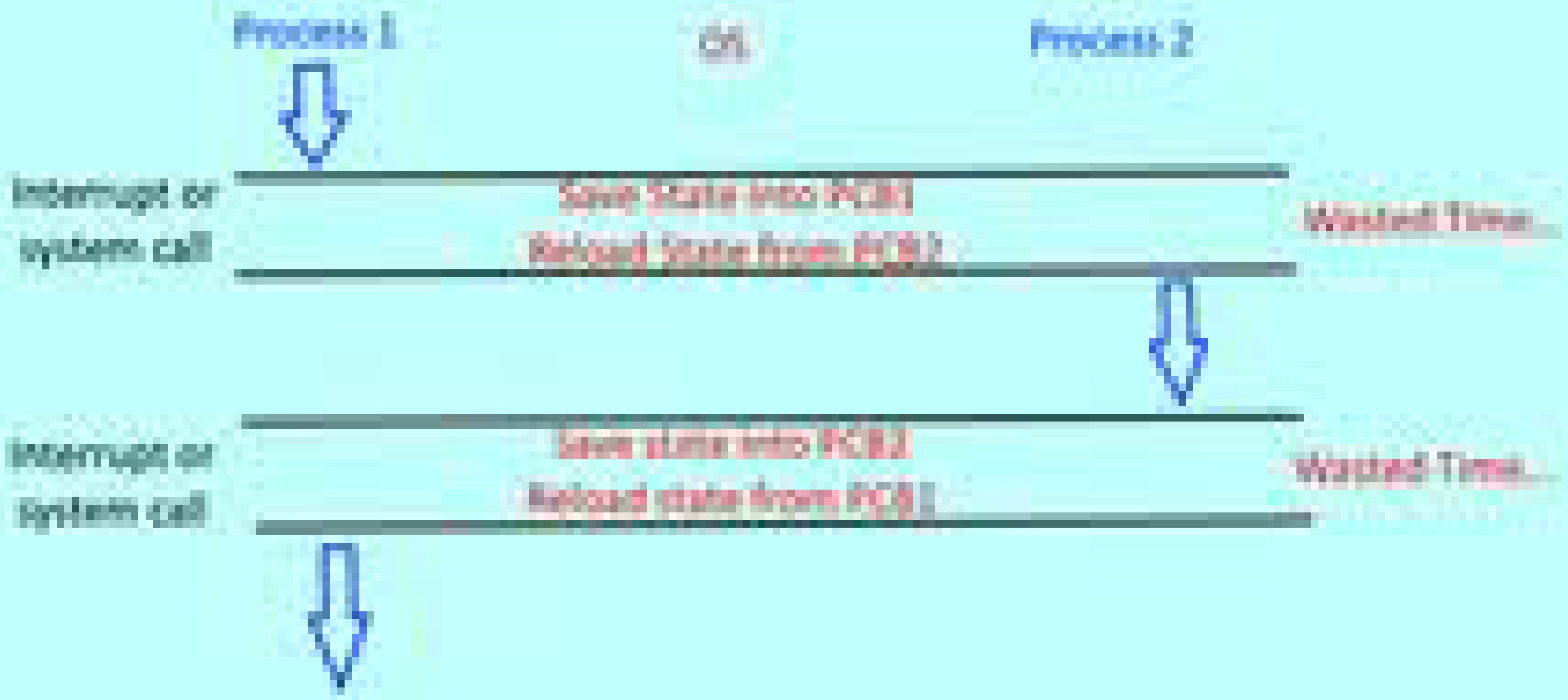
Context switching

- Is an essential feature of multithreading.
- A context switch is the process of storing the state of a process or of a thread, so that it can be restored and execution resumed from the same point later.
- This allows multiple processes to share a single CPU.
- It is the switching of CPU from one process or thread to another .

- Context switching in detail can be explained as the kernel performing the following activities with regard to processes on the CPU:
- (1) suspending the progression of one process and storing the CPU's state for that process somewhere in memory
- (2) retrieving the context of the next process from memory and restoring it in the CPU's register
- (3) returning to the location indicated by the program counter (i.e., returning to the line of code at which the process was interrupted) in order to resume the process.

Context Switch

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.



Multithreading -Processor Efficiency

ASSIGNMENT-02

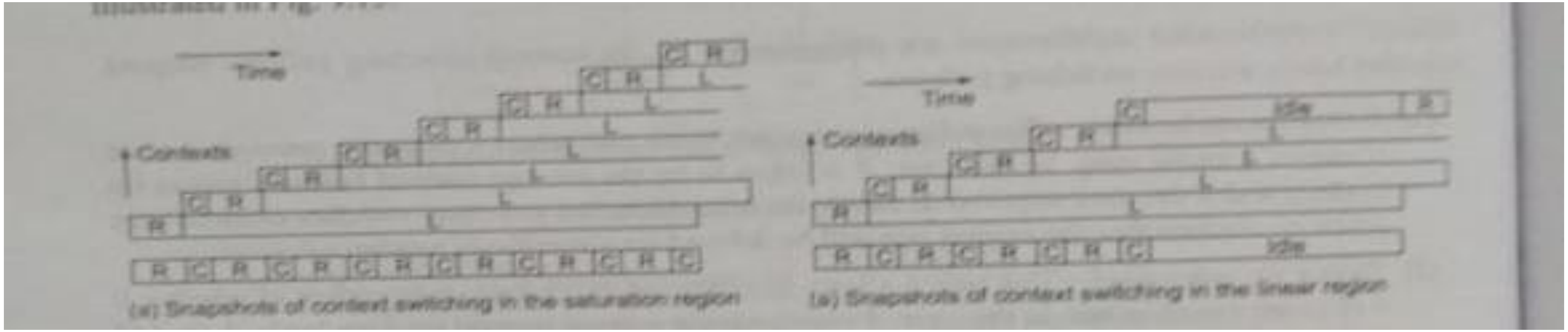
Devika Satheesh

Rollno:46

- No context switch and no switch overhead
- Efficiency of Single threaded machine is

$$E=R/R+L$$

- This shows performance degradation



(a) Snapshots of context switching in the saturation region

(b) Snapshots of context switching in the linear region

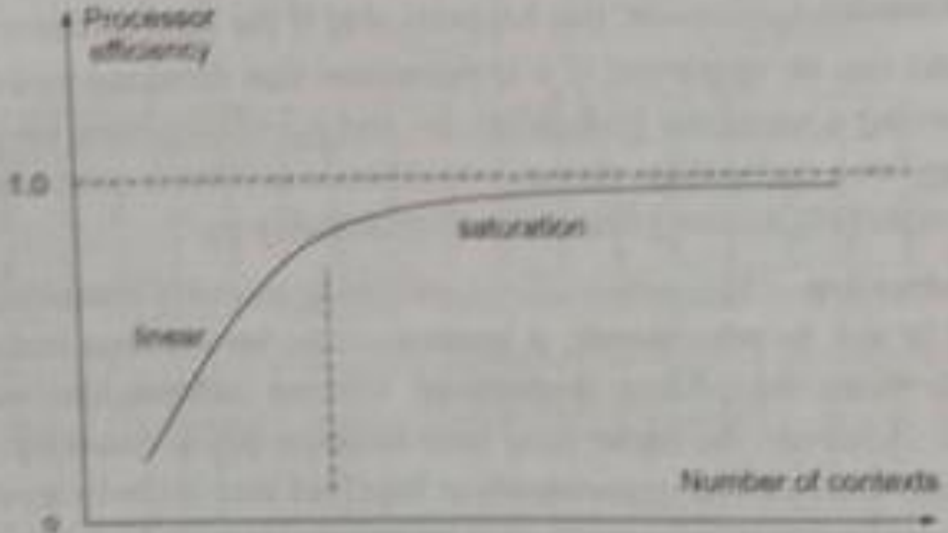
Saturation Region

- Processor operates with maximumutilization.
- Cycle of renewal process in this case is R+C and efficiency is:

$$E=R/R+C$$

- Observe that efficiency in saturation is independant of latency
- Saturation is achieved when(N-1)(R+C)>L.This gives saturation point under constant.

$$N=(L/R+C)+1$$

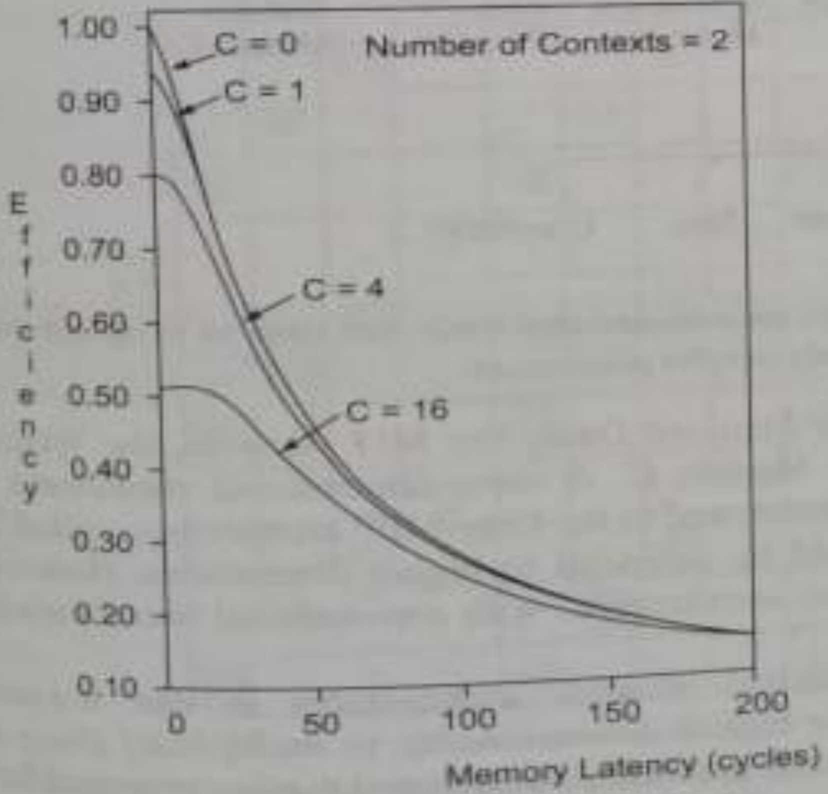


(c) Efficiency curve

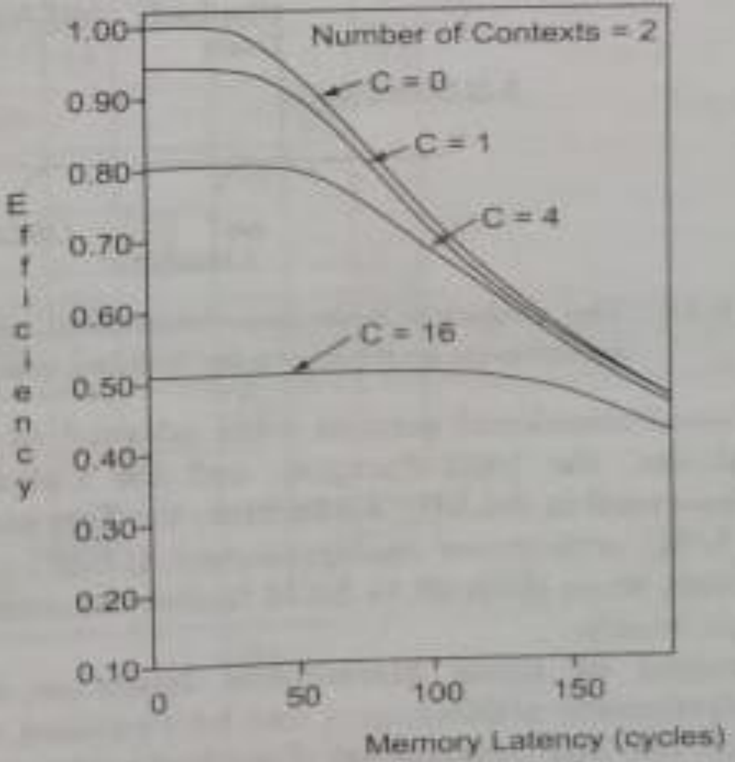
Linear Region

- When number of contexts is below saturation point,there may be no ready contexts which,so the procesor will experience idle cycles.
- The time required to switch to ready context,execute it until a remote reference is issued,and process the reference is equal to R+C+L.
- Assuming N is below saturation point,during this time all othercontexts have a turn in processor

$$E=NR/R+C+L$$



(a) Two contexts per processor



(b) Six contexts per processor

Fig. 9.16 Processor efficiency of a multithreaded architecture (Courtesy of R. Saavedra, D. E. Culler, and T. von Eicken, 1992)

MULTIDIMENSIONAL DATA ARCHITECTURE

POORNASREE R MOHAN

S7, CSE

ROLL NO:47

CONTENT:-

- Introduction of Multidimensional architecture
- Component of Multidimensional architectures
- Types of architectural models

  [A]. Data Cube Model.

  [B]. Star Schema Model.

  [C]. Snow Flake Schema Model.

  [D]. Fact Constellations.

INTRODUCTION

- The Multi- Dimensional Model was developed for implementing data warehouse and data marts
- It provide both a mechanism to store data and a way for computer data analysis.

COMPONENT OF MULTIDIMENSIONAL ARCHITECTURE

The two primary component of dimensional model are Dimensions and Facts.

- Dimensions:- Texture Attributes to analyses data.
- Facts:- Numeric volume to analyze business.

TYPES OF ARCHITECTURES

- [A]. Data Cube Model.
- [B]. Star Schema Model.
- [C]. Snow Flake Schema Model.
- [D]. Fact Constellations .

DATA CUBE MULTI-DIMENSIONAL ARCHITECTURE

- When data is grouped or combined together in multidimensional matrices called Data Cubes.
- In Two Dimension :- row & Column or Products &fiscal quarters.
- In Three Dimension:- one regions, products and fiscal quarters

STAR SCHEMA MODEL

- It is also known as Star Join Schema.
- It is the simplest style of data warehouse schema.
- It is called a Star Schema because the entity relationship
- diagram of this Schema resembles a star, with points
- radiating from central table.
- A star query is a join between a fact table and a no. of
- dimension table.
- Each dimension table is joined to the fact table using
- primary key to foreign key join but dimension table are
- not joined to each other.
- A typical fact table contain key and measure.

SNOW FLAKE SCHEMA

- It is slightly different from a star schema in which thedimensional tables from a star schema are organized into a hierarchy by normalizing them.
- The Snow Flake Schema is represented by centralized fact table which are connected to multiple dimensions.
- The Snow Flaking effecting only affecting the dimension tables and not the fact tables.

DIFFERENCE B/W STAR SCHEMA AND SNOW FLAKE:-

STAR SCHEMA

Star Schema dimension are De normalized with each dimension being represented in single table.

SNOW FLAKE SCHEMA

Snow flake Schema dimension are normalized into multiple related tables.

FACT CONSTELLATIONS

- It is set of fact tables that share some dimensions tables.
- It limits the possible queries for the data warehouse.

Wisconsin Multicube

- A large-scale, shared-memory multiprocessor architecture that uses a snooping cache protocol over a grid of buses.
- Each processor has a conventional (SRAM) cache optimized to minimize memory latency and a large (DRAM) snooping cache optimised to reduce bus traffic and to maintain consistency.
- Large snooping cache guarantee traffic on busses generated by I/O and access of shared data.

Programmer's view

- Multi-A set of processors having access to a common shared memory with no notion of geographical locality.
- Thus writing software, including the operating system, should be a straightforward extension of those techniques being developed for multis.

- The interconnection topology allows for a cache-coherent protocol for which most bus requests can be satisfied with no more than twice the number of bus operations required of a single-bus multi.
- The total symmetry guarantees that there are no topology-induced bottlenecks.
- The total bus bandwidth grows in proportion to the product of the number of processors and the average path length.

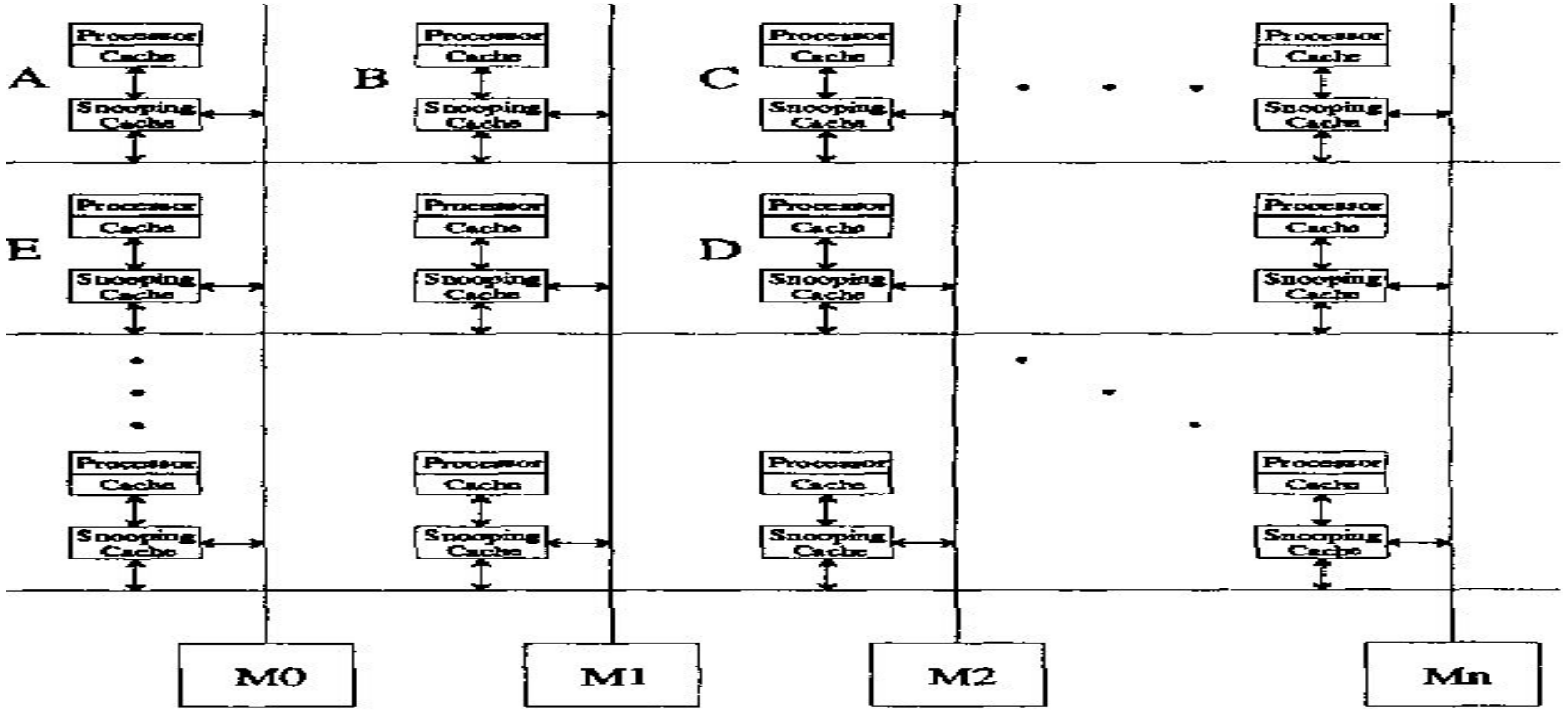


Figure 1: The Wisconsin Multicube

¹This cache is external to the processor, which likely will include an on-chip cache as well.

²Of course, the enormous total memory and computing power of this machine will permit

ORTHOGONAL MULTIPROCESSOR

Riya Binny
CSE S7 49

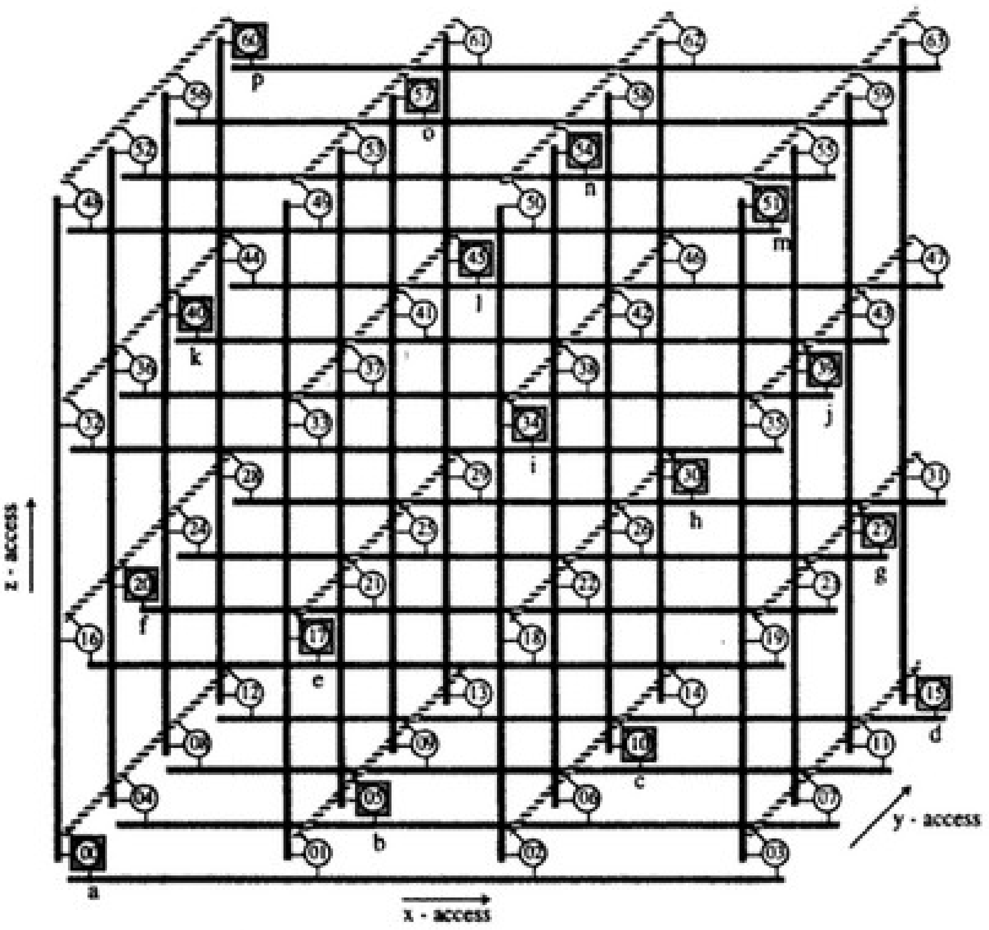
ORTHOGONAL MULTIPROCESSOR

- A generalised orthogonal multiprocessor is denoted as an OMP(n,k),where

  n = Dimension

  k = Mutliplicity
- There are p=k^(n-1) and m=k^n memory modules in the system where p>>n and p>>k.
- The system uses p memory buses,each spanning into n dimensions.But only one dimension is used in a given memory cycle.
- There are k memory modules attached to each spanning bus.

- The OMP architecture supports special purpose computations in which data sets can be regularly arranged as matrices.
- OMP is well suited for SPMD(single program, multiple data) operations in which n processors are synchronised at the memory access level when data sets are vectorized in matrix format.



(c) The 3-D OMP(3,4) architecture. (Processors are labeled a, b, ..., p; memory modules are labeled 00, 01, ..., 63)

OMP(n,k)	$p = k^{n-1}$	$m = k^n$
OMP(2, 8)	8	64
OMP(2, 16)	16	256
OMP(3, 8)	64	512
OMP(3, 16)	256	4096
OMP(4, 8)	512	4096
OMP(4, 16)	4096	65,536
OMP(5, 16)	65,536	1,048,576

Note: p = number of processors; m = number of memory modules.

- Each module is connected to n out of p buses through an n-way switch.
- Dimension n corresponds to the number of accessible ports that each memory module has.
- Each memory module is shared by n out of p=k^(n-1) processors.

Multidimensional extensions

ROBIN JOSEPH

S7 CSE

ROLL NO : 50

Multidimensional extensions

Used to :

- Identify dimensions by which objects are perceived or evaluated.
- Position the objects with respect to those dimensions.
- Make positioning decisions for new and old products.

Basic concepts of Multidimensional scaling (MDS)

- MDS uses proximities among different objects as input.
- Proximities data is used to produce a geometric configuration of points in a two-dimensional space as output.
- The fit between the derived distances and the two proximities in each dimension is evaluated through a measure called stress.
- The approximate number of dimensions required to locate objects can be obtained by plotting stress values against the number of dimensions.

Attribute based MDS

Advantages:

- Attributes can have diagnostic and operational values.
- Attribute data is easier for the respondents to use.

Disadvantages:

- If the list of attributes is not accurate and complete, the study will suffer.
- The respondents may not perceive or evaluate objects in terms of underlying attributes.

Application of MDS with non-attribute data

- Reflect the perceived similarity of two objects from the respondents perspectives.
- Perpectual map is obtained from the average similarity ratings.
- Able to find the smallest number of dimensions for which there is a reasonably good fit between the input similarity rankings and the rankings of the distance between objects in the resulting space.

MIT J-Machine - Architecture and MDP Design

Roshin Jojo

S7 CSE 52

Introduction

- The J–Machine (Jellybean-Machine) was a parallel computer designed by the MIT Concurrent VLSI Architecture group in conjunction with the Intel Corporation. The machine used "jellybean" parts—cheap and multitudinous commodity parts, each with a processor, memory, and a fast communication interface—and a novel network interface to implement fine grained parallel programs

Architecture

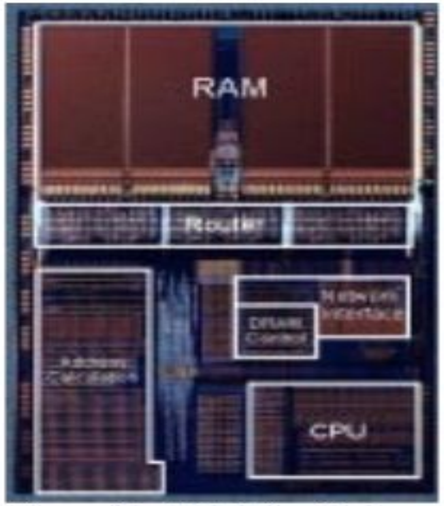


Figure 1: MDP Die Photo

Figure 2: J-Machine

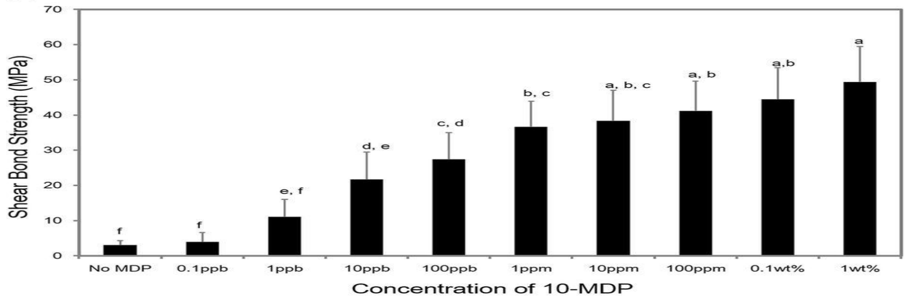
The J-Machine demonstrated the use of a jellybean part, a commodity part incorporating a processor, memory, and a fast communication interface, as a building block for computing systems. It was a fine-grain parallel computer designed to exploit large amounts of parallelism by balancing the use of silicon area between processor and memory.

The J-Machine provided a small set of efficient communication and synchronization mechanisms that were used to support a broad range of programming models. It also provided fast user-to-user messaging without software intervention by having each message dispatch a message handler.

MDP Design

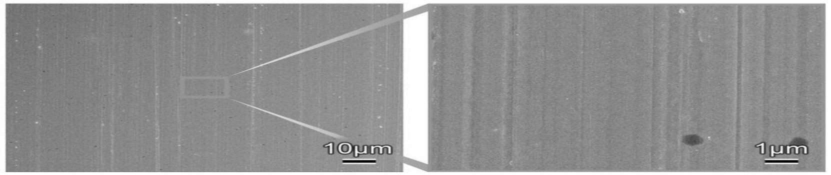
Each processing node of the J-Machine is composed of a single Message-Driven Processor (MDP) chip and three 1M 4 DRAM chips. The resulting node is 2" by 3". Each J-Machine processor board contains 64 MDP chips along with their external memories. The board measures 26" by 20.5". Each pair of nodes shares a set of elastomeric con- ductors to communicate with the corresponding nodes on the boards above and below in the board stack. A total of 48 elastomeric connectors held in four connector hold- ers provide 1420 electrical connections between adjacent boards. Of these connections, 960 are used for signalling and the remaining 460 are ground returns. The use of elastomeric conductors enables the J-Machine to achieve a very high network bisection bandwidth (a peak rate of 14.4 Gbits/sec) in a very small cross sectional area (2 ft 2) and at the same time keep the length of the longest channel wire in a 1024-node machine under four inches. A stack of sixteen processor boards contains 1024 processing nodes (12,500 MIPS peak) and 1Gbyte of DRAM memory in a volume of 4 ft 3 , and dissipates 1500W (1.5W per node).

(A)



Shear Bond Strength (MPa)
70
60
50
40
30
20
10
0
f
f
e, f
d, e
c, d
b, c
a, b, c
a, b
a,b
a
No MDP
0.1ppb
1ppb
10ppb
100ppb
1ppm
10ppm
100ppm
0.1wt%
1wt%
Concentration of 10-MDP

(B)



10μm
1μm

THANKS!

MIT J-Machine - Instruction Set Architecture and Communication Support

Sara Jacob
S7 CSE 53

Instruction-Set Architecture

- The MDP extended a conventional microprocessor instruction-set architecture with instructions to support parallel processing.
- The instruction set contained fixed-format,three address instructions.
- Two 17-bit instructions fit into each 36-bit word with 2 bits reserved for type checking.

- Separate register sets were provided to support rapid switching among 3 execution levels

    1) background

    2) priority 0 (P0)

    3) priority 1 (P1)

- MDP executed at background level while no message created a task, and initiated execution upon message arrival at P0 & P1 level depending on message priority.
- P1 level had higher priority than P0 level.

Communication Support

- MDP transmitted a message using a series of SEND instruction.

- For sending a four-word message using 3 variants of SEND instruction

```
SEND     R0,0           ;  send net address
SEND2    R1,R2,0        ;  header and receiver
SEND2E  R3,[3,A3],0   ;  selector and communication end  message
```

- First SEND instruction reads the absolute address of the destination node in <X,Y,Z> format from R0 and forwards it to the network hardware.
- SEND2 reads first 2 words of the message out of registers R1 and R2 and enqueues them for transmission.
- Final instruction enqueues 2 additional words of data, one from R3 and one from memory.

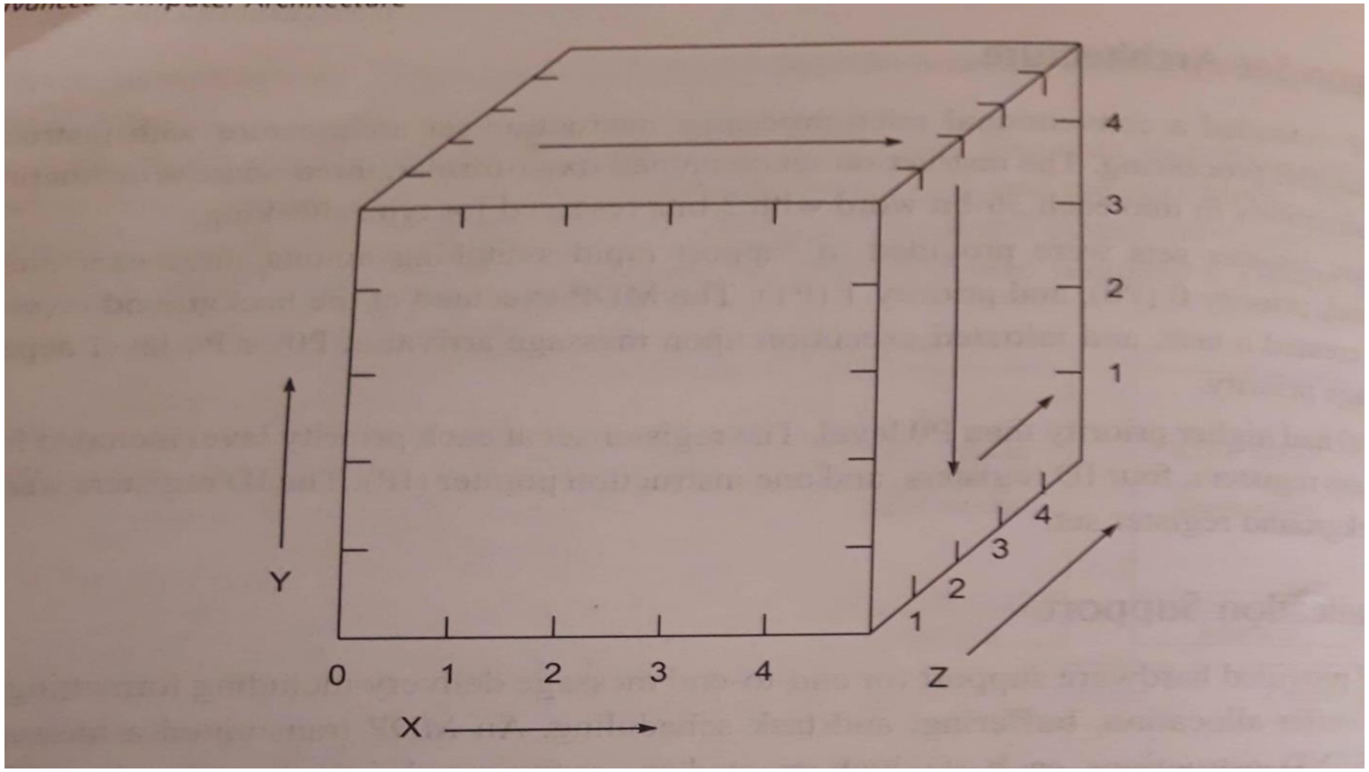
MIT J-machine Message Format and routing

Sherin George

Roll no : 54

- The J-machine used deterministic dimension-order **E-cube routing.**
- Here all messages are routed first in the x-dimension, then in the y-dimension, and then in the z-dimension.
- Since messages routed in dimension order and messages running in opposite directions along the same direction cannot block, resource cycles are avoided, making the network **deadlock-free**

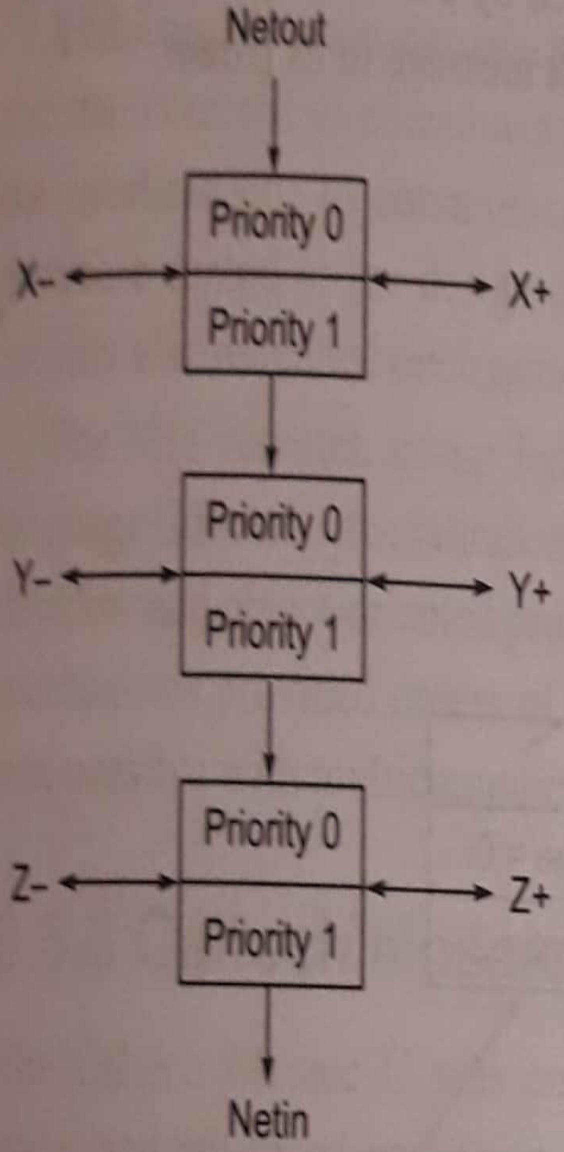
- The below figure shows E-cube routing from node(1,5,2) to node(5,1,4) on a 6-ary 3-cube.



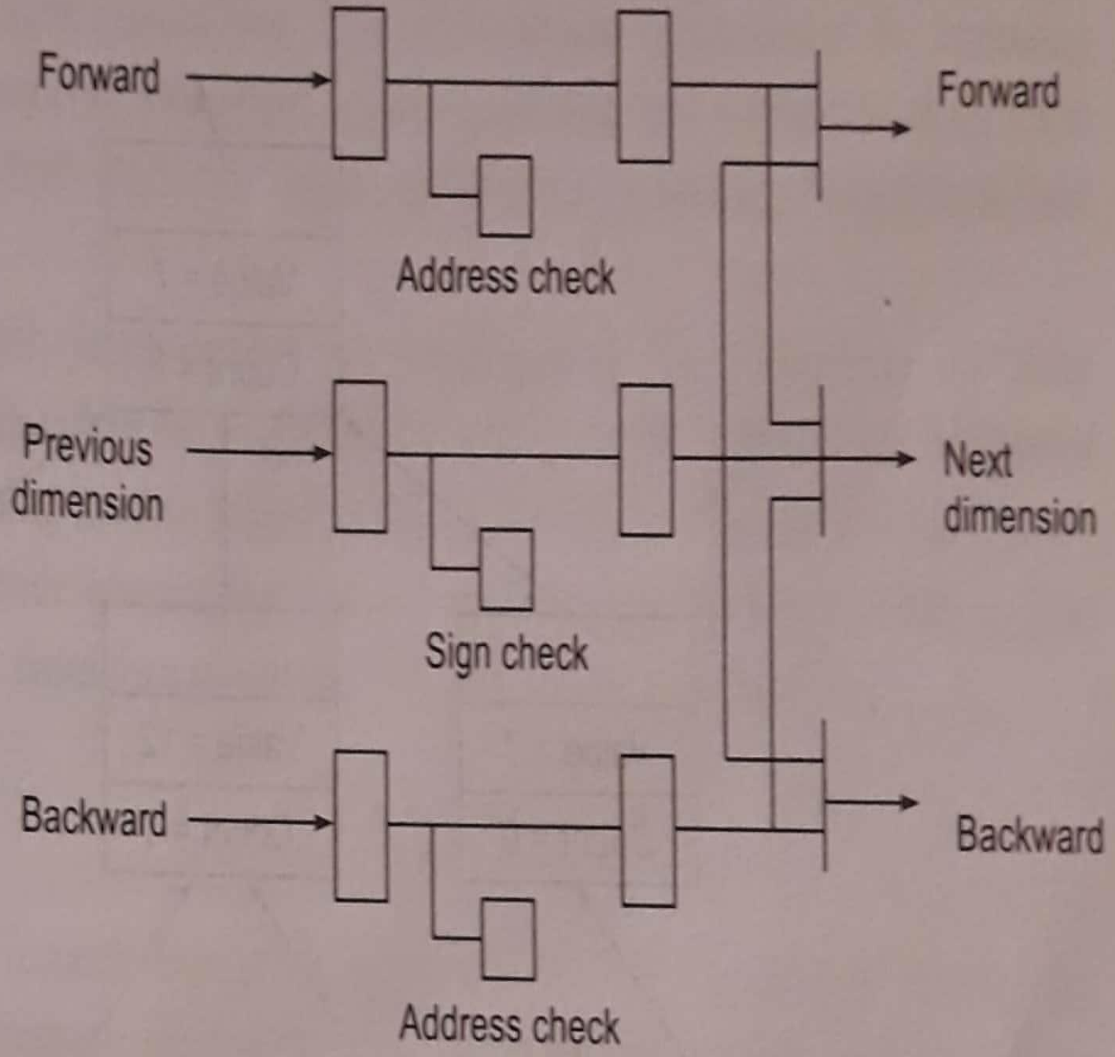
Router Design

- The routers formed the switches in a MIT J-machine network and delivered messages to their destinations.
- It contain three independent routers, one for each bidirectional dimension of the network.
- Each router contains two separate virtual networks with different priorities that shared the same physical channels.

- The priority-1 network could preempt the wires even if the priority-0 network was congested or jammed.
- Each of the 18 router paths contained buffers, comparators, and output arbitration.
- A message entering the dimension competed with messages continuing in the dimension at a two-to-one switch.



(a) Dual-priority levels per dimension in the router



(a) Each priority with forward, reverse, and previous data paths to the next dimension.

- Two priorities of messages shared the physical wires but used completely separate buffers and routing logic.
- This allowed priority-1 messages to proceed through blockages at priority 0.

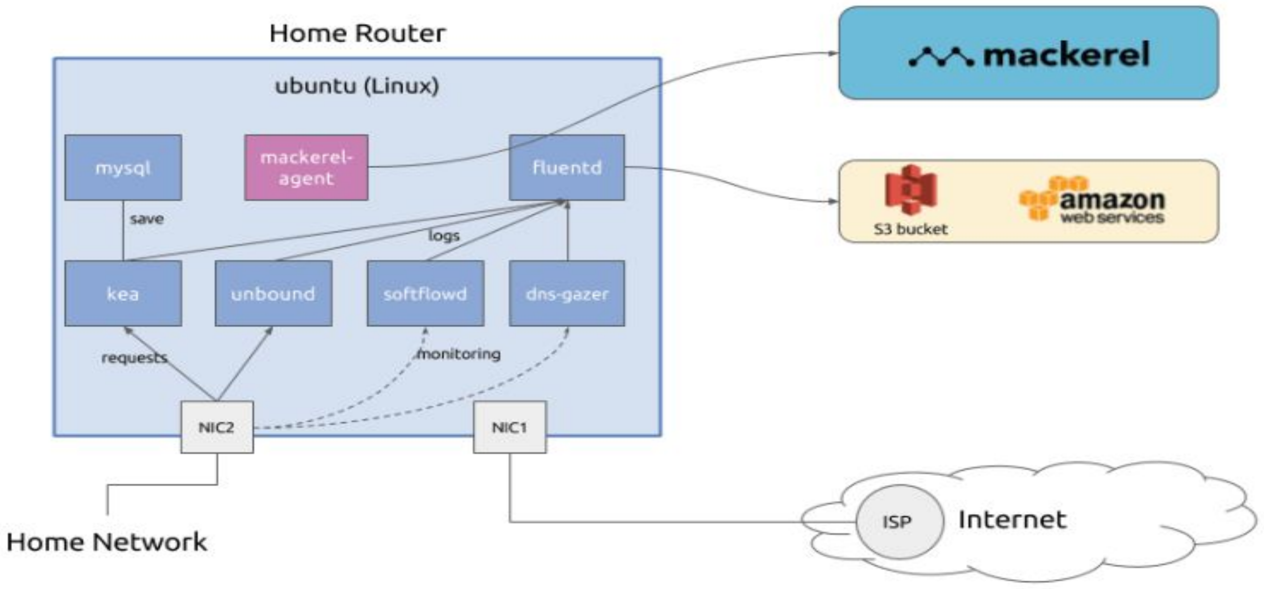
MIT J-machine - Router Design, Synchronization and Research Issues

Sreehari Rajeev
CSE,S7,55

INTRODUCTION

- The J-machine project was started at MIT in about 1988 as an experiment in message-passing computing based on work that Bill Dally did at Caltech for his doctoral dissertation.
- The work was driven by the VLSI philosophy "processors are cheap" and "memory is expensive". This philosophy is based on a idealistic view of VLSI economics, in which the cost of a function is based on the VLSI area dedicated to it.
- The design of the J-machine incorporated several novel technologies.

ROUTER DESIGN



SYNCHRONIZATION

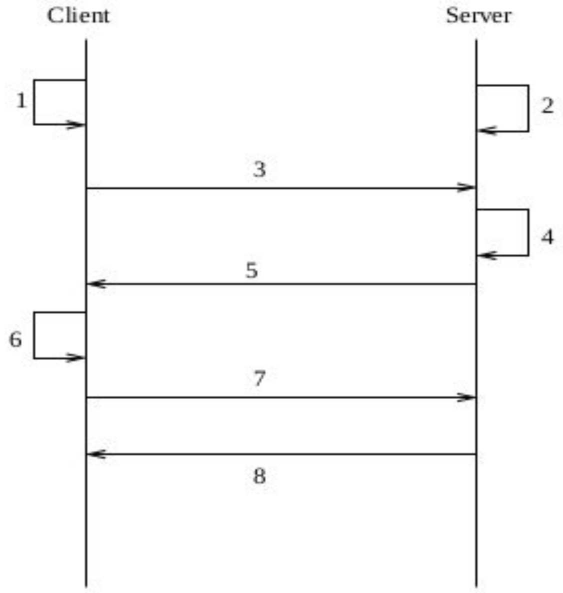


Figure 2: Simplified diagram of the synchronization

RESEARCH ISSUES

- J machine is an exploratory research project.
- Rather than being specialized in a single model of computation, the MDP incorporates primitive mechanisms for effective communications, synchronization and naming.
- The machine is used as a platform for software experiments in fine-grain parallel programming.

THANK YOU…

Caltech Mosaic C

SREELAKSHMI S.K

S7 , CSE

ROLL NO : 56

INTRODUCTION

- The Caltech Mosaic C is an experimental, fine-grain multicomputer that employs single-chip nodes and advanced packaging technology to demonstrate the performance/cost advantages of the fine-grain-multicomputer architecture.

ARCHITECTURE

- Each Mosaic node includes 64KB of single-clock-cycle dynamic RAM
- 2KB of self-test and bootstrap ROM
- An 11MIPS processor
- A packet interface
- A 60MB/s, two-dimensional, self-timed router

OPERATIONS

- The operating node is a single, 9.25mm\Theta10.00mm, 1.2 m-feature-size, CMOS chip
- CMOS chip operate at V dd = 5V
- operates at 30MHz and dissipates 0.5W
- These chips are packaged by tape-automated bonding in 8\Theta8 arrays on circuit boards that can, in turn, be composed in two dimensions to construct arbitrarily large arrays of nodes

PROGRAMMING FEATURES

- The Mosaic programming system consists of a compiler for a source programming notation called C + -,, and a distributed runtime system.
- C + - is an extension of C ++ that supports concurrent processes in much the same way that C ++ supports program objects
- The runtime system, which is written in C + -, provides copy less message handling, highly distributed resource management, automatic process placement, and scheduling.

APPLICATIONS

- The Mosaic is intended both for large-scale-computing and for embedded- systems applications.
- A 16K-node Mosaic system is being constructed at Caltech to explore the programmability and application span of the architecture for large-scale computations.
- The ATOMIC local-area network is an early example of an embedded-system application of Mosaic components.

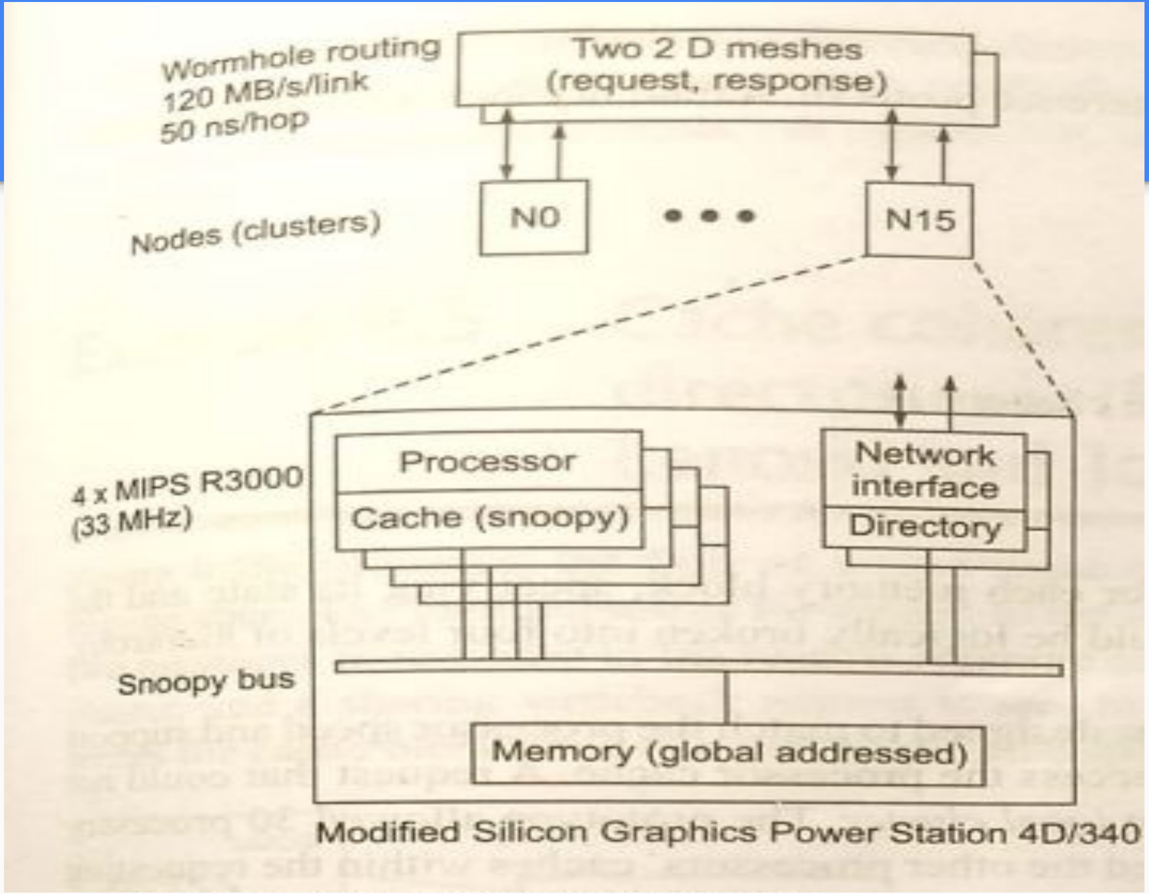
Stanford Dash Multiprocessor Prototype Architecture

Submitted by
V J HARAN

Main Features

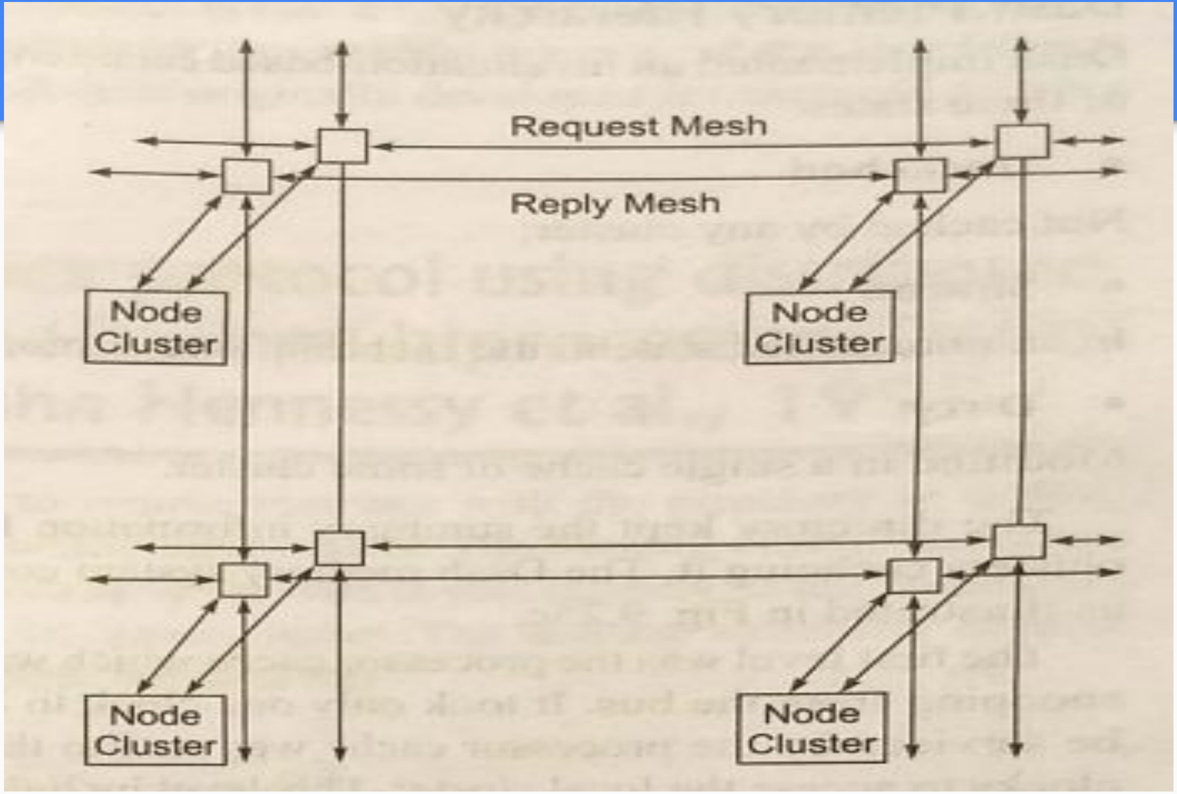
- Developed by John Hennessey and co-workers at Stanford University,1988
- DASH(Directory Architecture for Shared Memory)
- A scalable high-performance machine
- Single address space, coherent caches & distributed memories
- Incorporates upto 64 MIPS R3000/R3010 microprocessors with 16 clusters of 4 PEs each
- Cluster hardware : Silicon Graphics 4D/340 nodes
- Pair of Wormhole routed mesh networks are used for interconnection among 16 multiprocessor clusters
- Channel width : 16 bits
- System is scalable to support 100's of 1000's of processors

Prototype node implementation

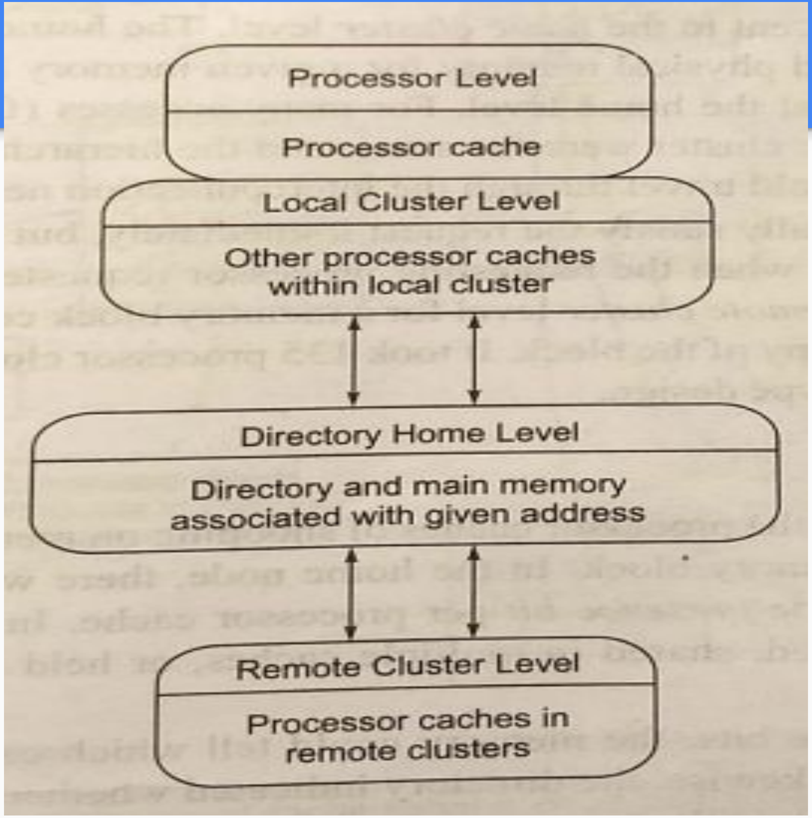


- To use the 4D/340 in the Dash ---
    1. Designed new board to support intercluster interface and directory memory
    2. Modified central bus arbiter to accept mask from directory
- Mask held off a processor's retry until a remote request was serviced
- New directory control board contains ---
    - Directory memory
    - Intercluster coherence state machines and buffers
    - Local section of global interconnection network
- Permitted incremental porting
- Factors for improved performance includes mechanisms for reducing and tolerating latency, well-designed I/O capabilities

Block diagram of 2x2 mesh interconnect



Logic Memory Hierarchy


Processor Level
Processor cache
Local Cluster Level
Other processor caches within local cluster
Directory Home Level
Directory and main memory associated with given address
Remote Cluster Level
Processor caches in remote clusters

THE DATA FLOW MODEL (OF A COMPUTER)

- Von Neumann model: An instruction is fetched and executed in control flow order
  - As specified by the instruction pointer
  - Sequential unless explicit control flow instruction

- Dataflow model: An instruction is fetched and executed in data flow order
  - i.e., when its operands are ready
  - i.e., there is no instruction pointer
  - Instruction ordering specified by data flow dependence
    - Each instruction specifies "who" should receive the result
    - An instruction can "fire" whenever all operands are received
  - Potentially many instructions can execute at the same time
    - Inherently more parallel

CSA ASSIGNMENT 2
Evolution of Dataflow Computers, Dataflow Graphs and Static vs Dynamic Dataflow

Vysakh Prasannan

S7,CSE

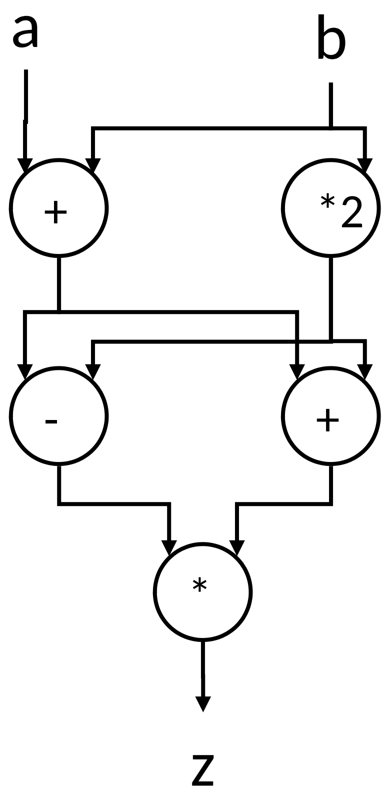
Roll No:59

VON NEUMANN VS DATAFLOW

- Consider a Von Neumann program
  - What is the significance of the program order?
  - What is the significance of the storage locations?

```
v <= a + b;
w <= b * 2;
x <= v - w
y <= v + w
z <= x * y
```

Sequential

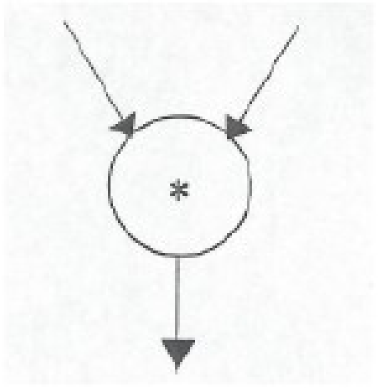


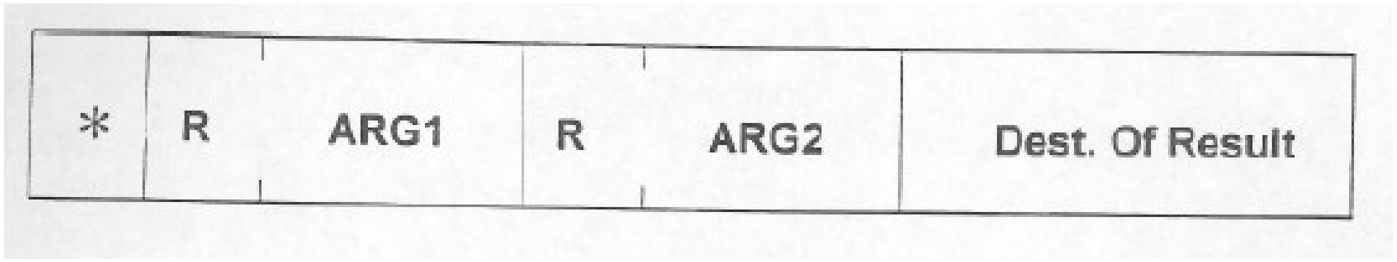
Dataflow

- Which model is more natural to you as a programmer?

MORE ON DATA FLOW

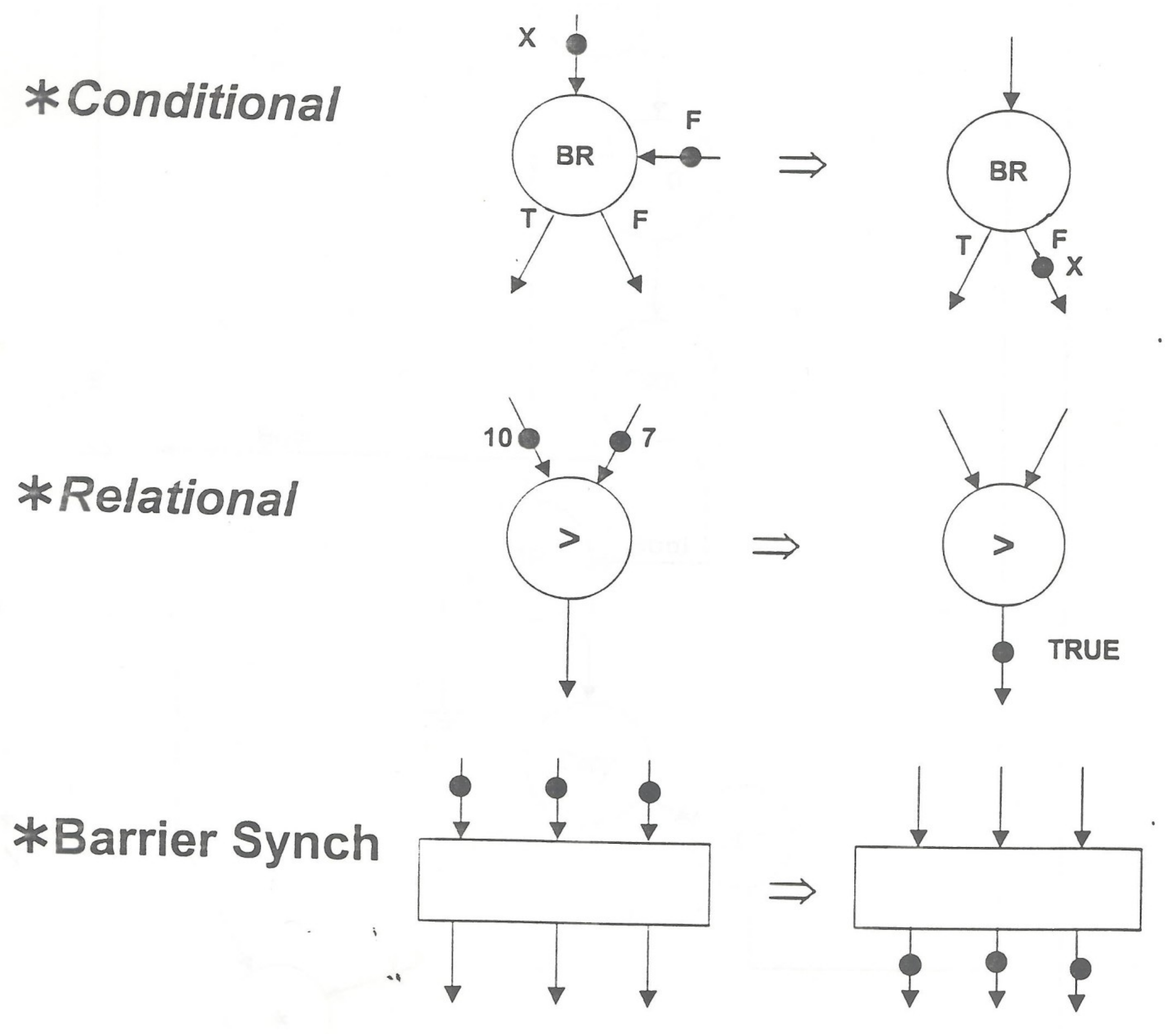
- In a data flow machine, a program consists of data flow nodes
  - A data flow node fires (fetched and executed) when all it inputs are ready
    - i.e. when all inputs have tokens
- Data flow node and its ISA representation



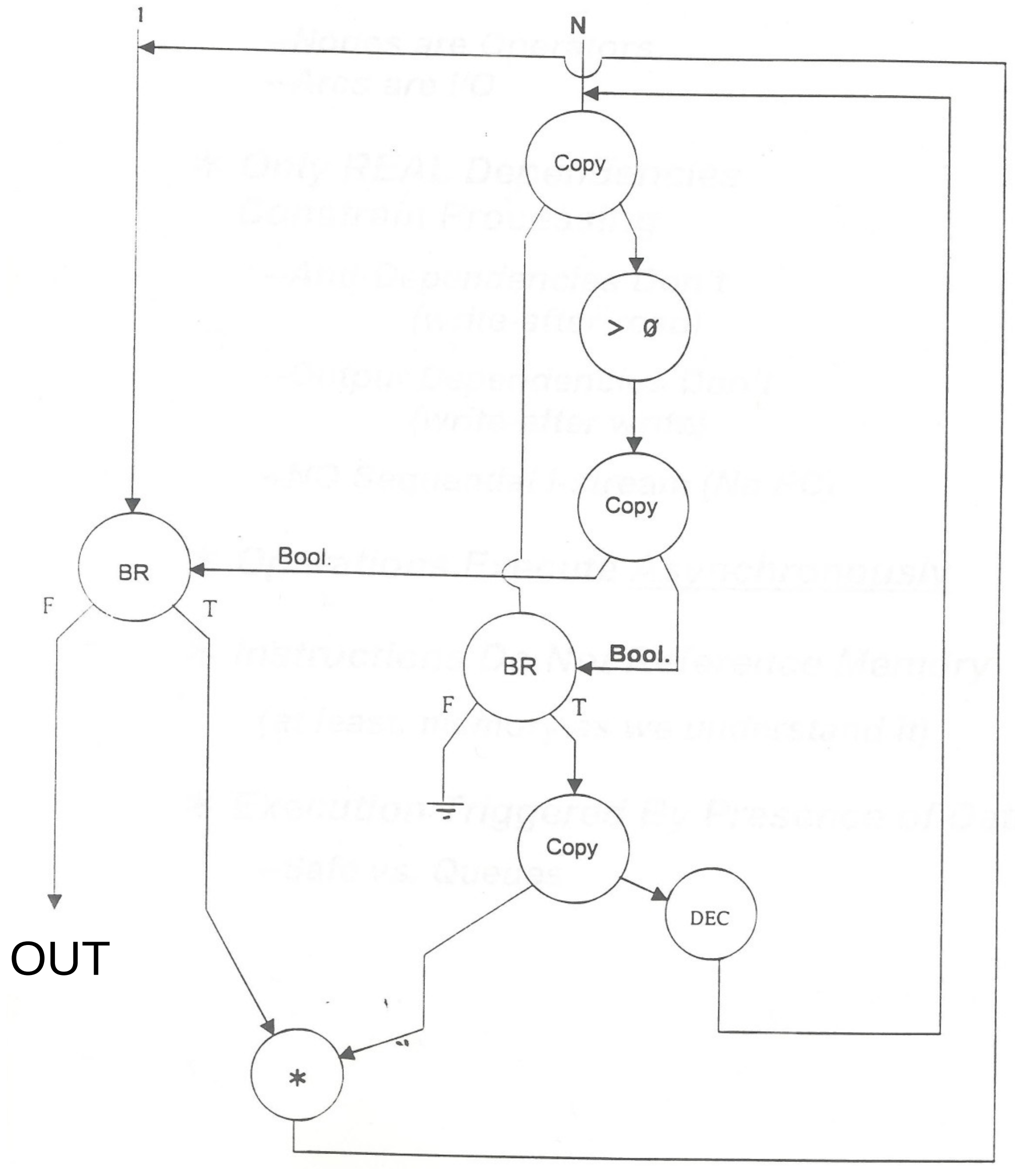


*	R	ARG1	R	ARG2	Dest. Of Result

DATA FLOW NODES



AN EXAMPLE DATA FLOW PROGRAM



CONTROL FLOW VS. DATA FLOW

a := x + y
b := a × a
c := 4 — a

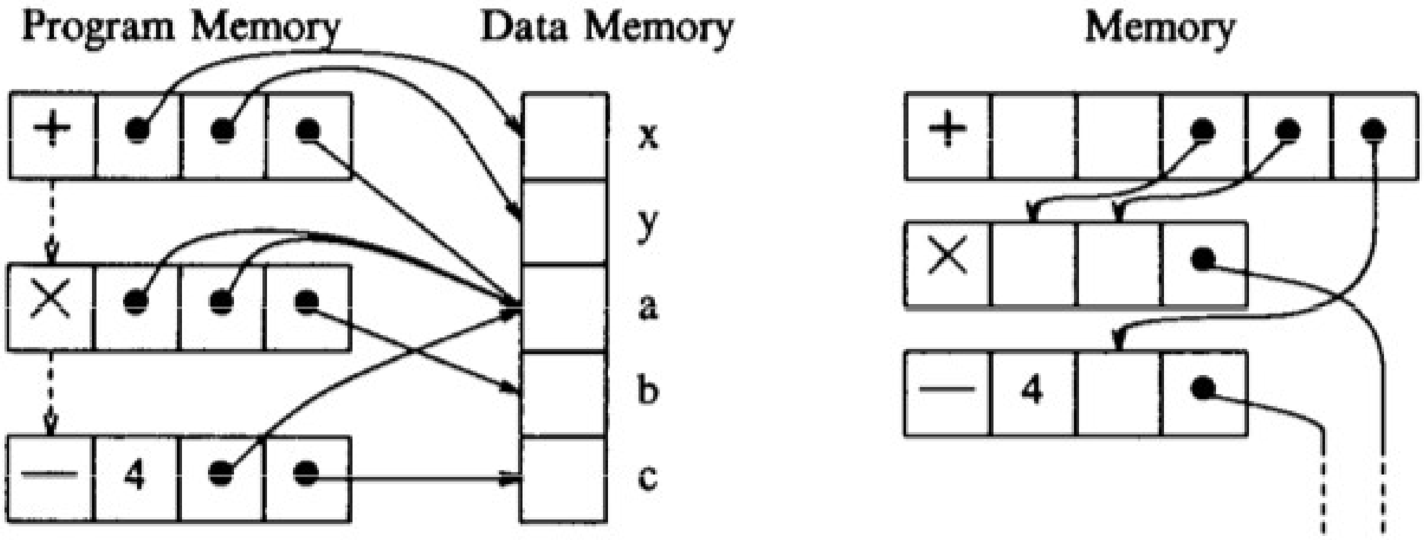


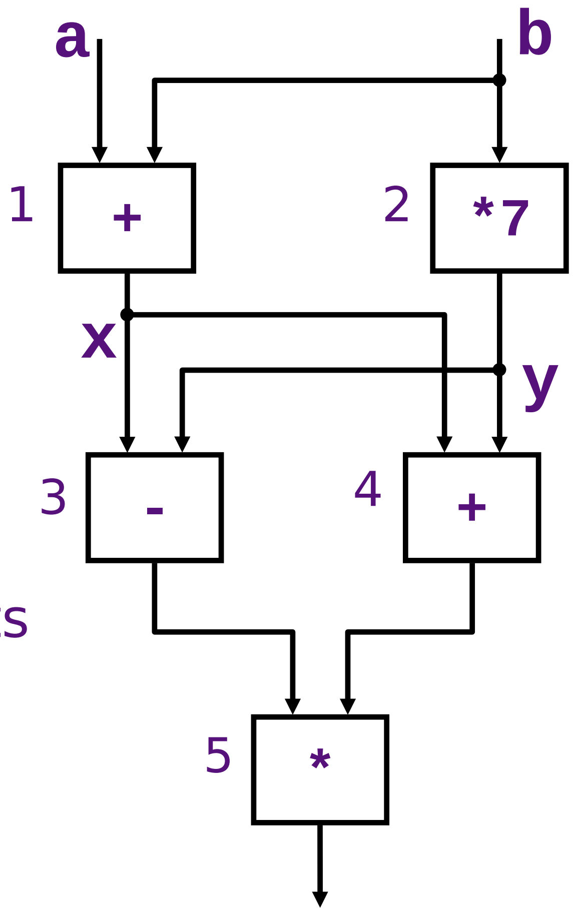
Figure 2. A comparison of control flow and dataflow programs. On the left a control flow program for a computer with memory-to-memory instructions. The arcs point to the locations of data that are to be used or created. Control flow arcs are indicated with dashed arrows; usually most of them are implicit.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

Dataflow Machine:
Instruction Templates

	Opcode	Destination 1	Destination 2		Operand 1		Operand 2
1	+	3L	4L	🟨		🟨	
2				🟨		🟨	
3	* -	3R 5L	4R	🟨		🟨	
4	+	5R		🟨		🟨	
5	*	out		🟨		🟨	

Presence bits

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



Dennis and Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," ISCA 1974.

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

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

ANOTHER WAY OF EXPLOITING PARALLELISM

- SIMD:
  - Concurrency arises from performing the same operations on different pieces of data

- MIMD:
  - Concurrency arises from performing different operations on different pieces of data
    - Control/thread parallelism: execute different threads of control in parallel ✉ multithreading, multiprocessing
  - Idea: Use multiple processors to solve a problem

FLYNN'S TAXONOMY OF COMPUTERS

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

SIMD PROCESSING

- Concurrency arises from performing the same operations on different pieces of data
  - Single instruction multiple data (SIMD)
  - E.g., dot product of two vectors

- Contrast with thread ("control") parallelism
  - Concurrency arises from executing different threads of control in parallel

- Contrast with data flow
  - Concurrency arises from executing different operations in parallel (in a data driven manner)

- SIMD exploits instruction-level parallelism
  - Multiple instructions concurrent: instructions happen to be the same

SIMD PROCESSING

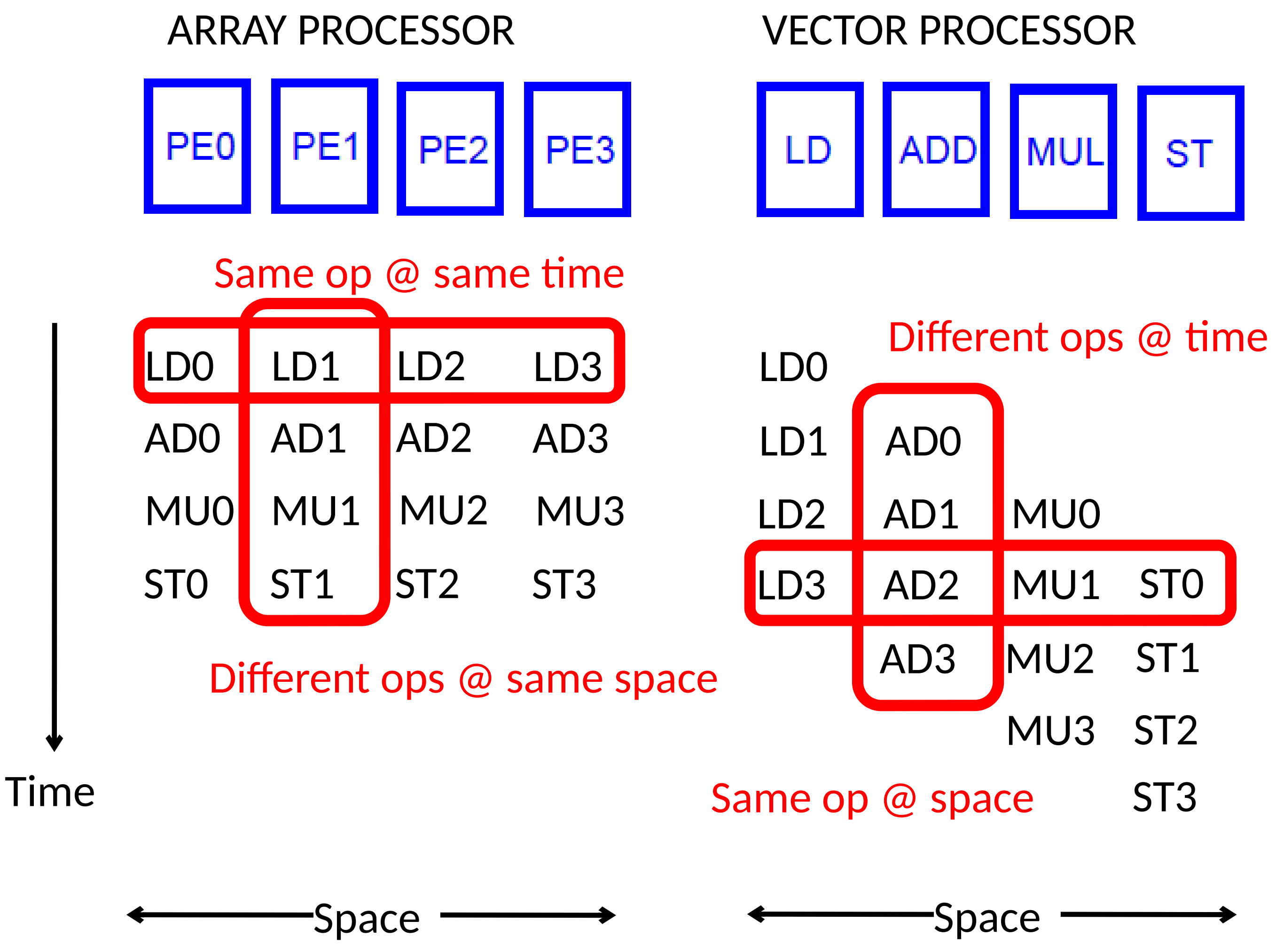
- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements

- Time-space duality
  - Array processor: Instruction operates on multiple data elements at the same time
  - Vector processor: Instruction operates on multiple data elements in consecutive time steps

ARRAY VS. VECTOR PROCESSORS

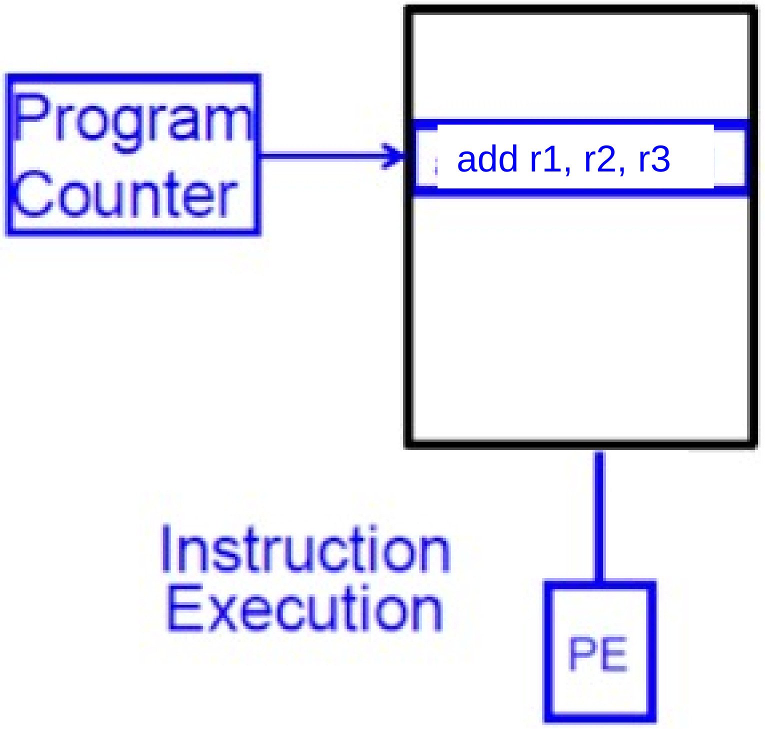
Instruction Stream

LD	VR A[3:0]	
ADD	VR VR, 1	
MUL	VR VR, 2	
ST	A[3:0] VR	



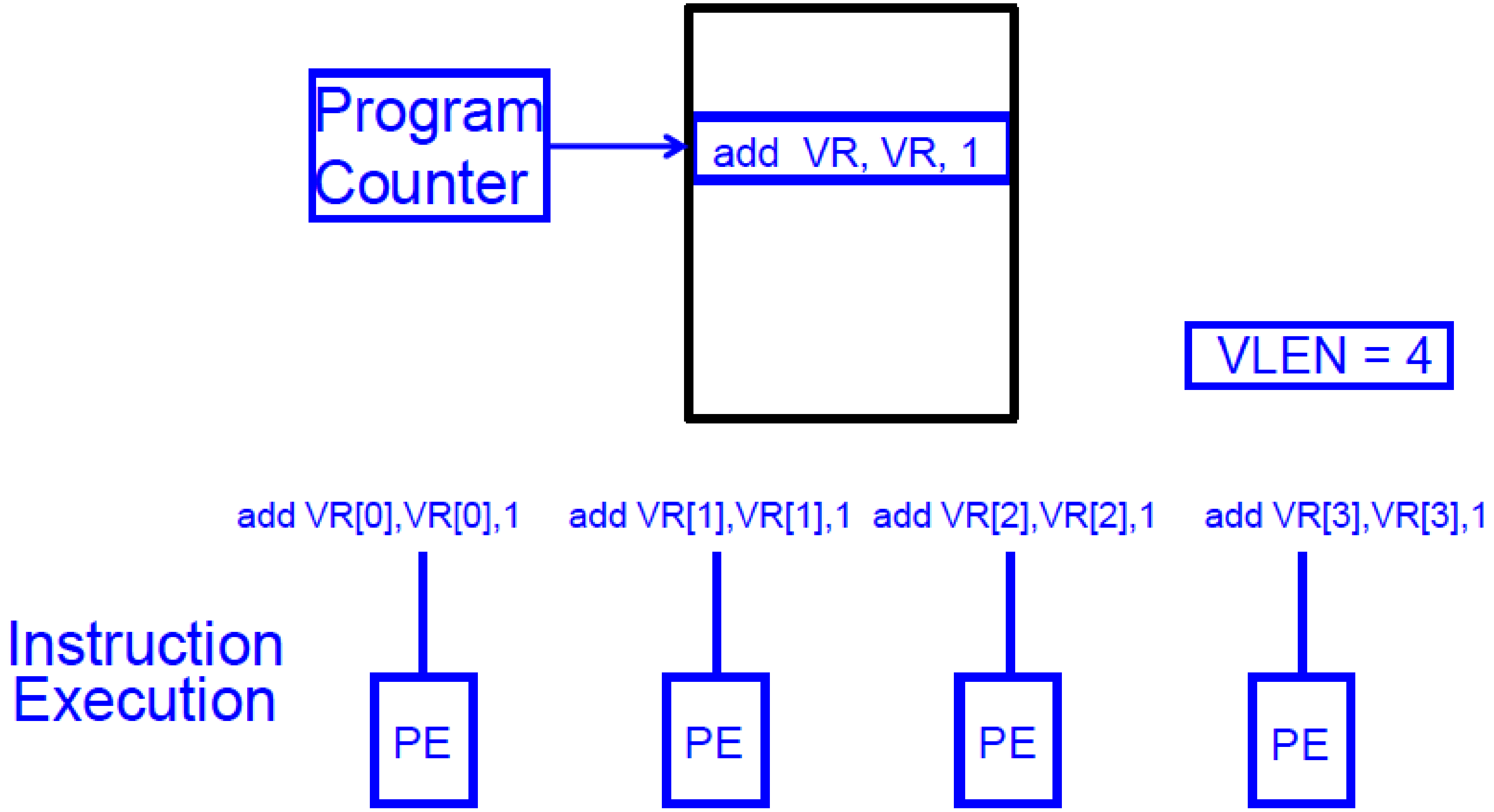
SCALAR PROCESSING

- Conventional form of processing (von Neumann model)



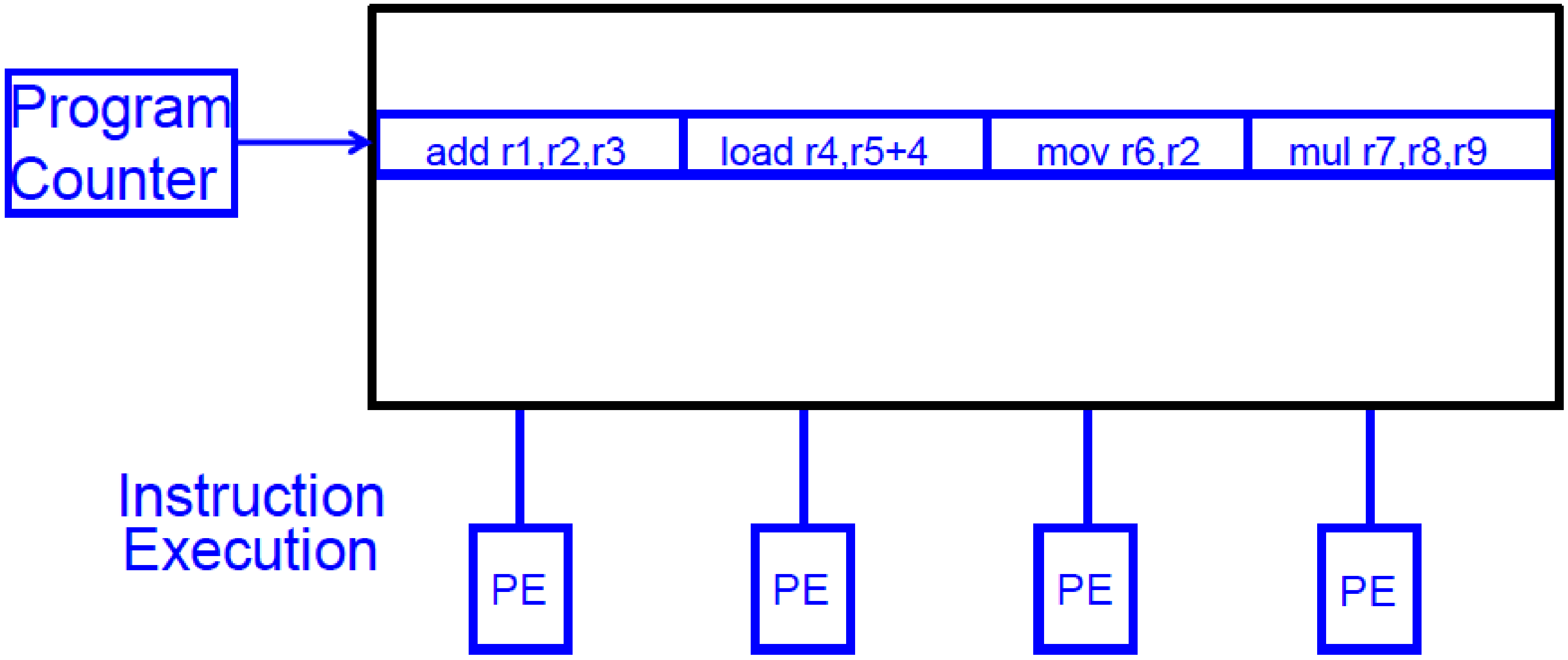
SIMD ARRAY PROCESSING

- Array processor



VLIW PROCESSING

- Very Long Instruction Word
  - We will get back to this later



VECTOR PROCESSORS

- A vector is a one-dimensional array of numbers
- Many scientific/commercial programs use vectors

```
for (i = 0; i<=49; i++)
   C[i] = (A[i] + B[i]) / 2
```

- A vector processor is one whose instructions operate on vectors rather than scalar (single data) values
- Basic requirements
  - Need to load/store vectors ✉ vector registers (contain vectors)
  - Need to operate on vectors of different lengths ✉ vector length register (VLEN)
  - Elements of a vector might be stored apart from each other in memory ✉ vector stride register (VSTR)
    - Stride: distance between two elements of a vector

VECTOR PROCESSOR ADVANTAGES

+ No dependencies within a vector
  - Pipelining, parallelization work well
  - Can have very deep pipelines, no dependencies!

+ Each instruction generates a lot of work
  - Reduces instruction fetch bandwidth

+ Highly regular memory access pattern
  - Interleaving multiple banks for higher memory bandwidth
  - Prefetching

+ No need to explicitly code loops
  - Fewer branches in the instruction sequence

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

Dynamic Dataflow

- 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

Static versus Dynamic Dataflow Machines

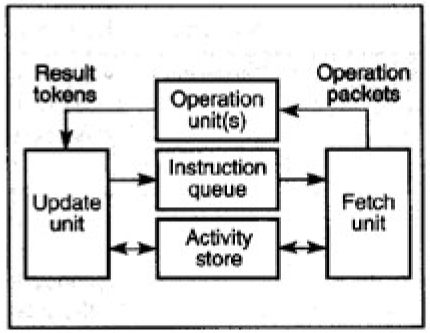


Figure 1. The basic organization of the static dataflow model.

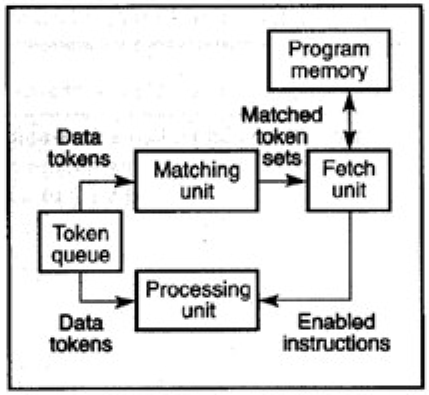
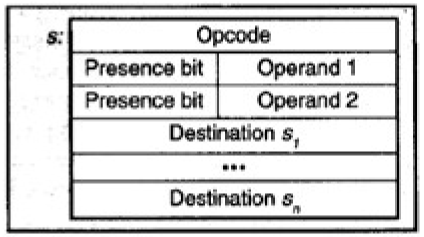


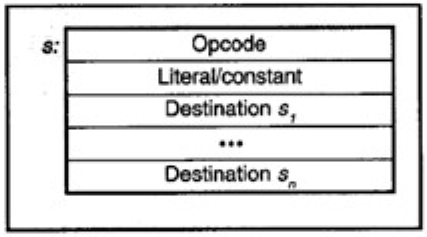
Figure 3. The general organization of the dynamic dataflow model.



s:

Opcode	
Presence bit	Operand 1
Presence bit	Operand 2
Destination s_1	
•••	
Destination s_n	

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



s:

Opcode
Literal/constant
Destination s_1
•••
Destination s_n

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

Pure Dataflow Machines, Explicit Token Store Machines and hybrid and Unified Architectures

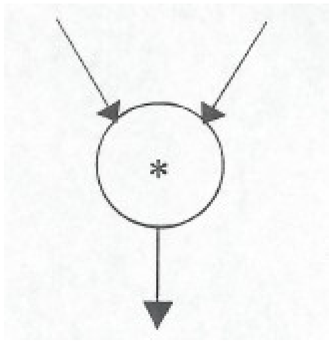
YADHU KRISHNAN P D

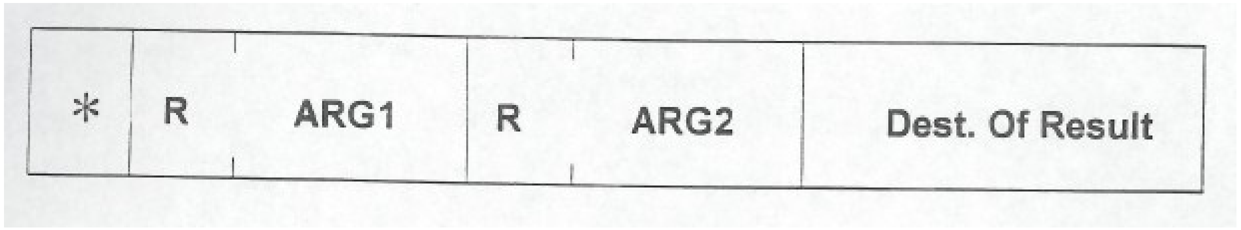
S7 CSE

ROLL NO : 60

Pure Dataflow Machines

- 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:



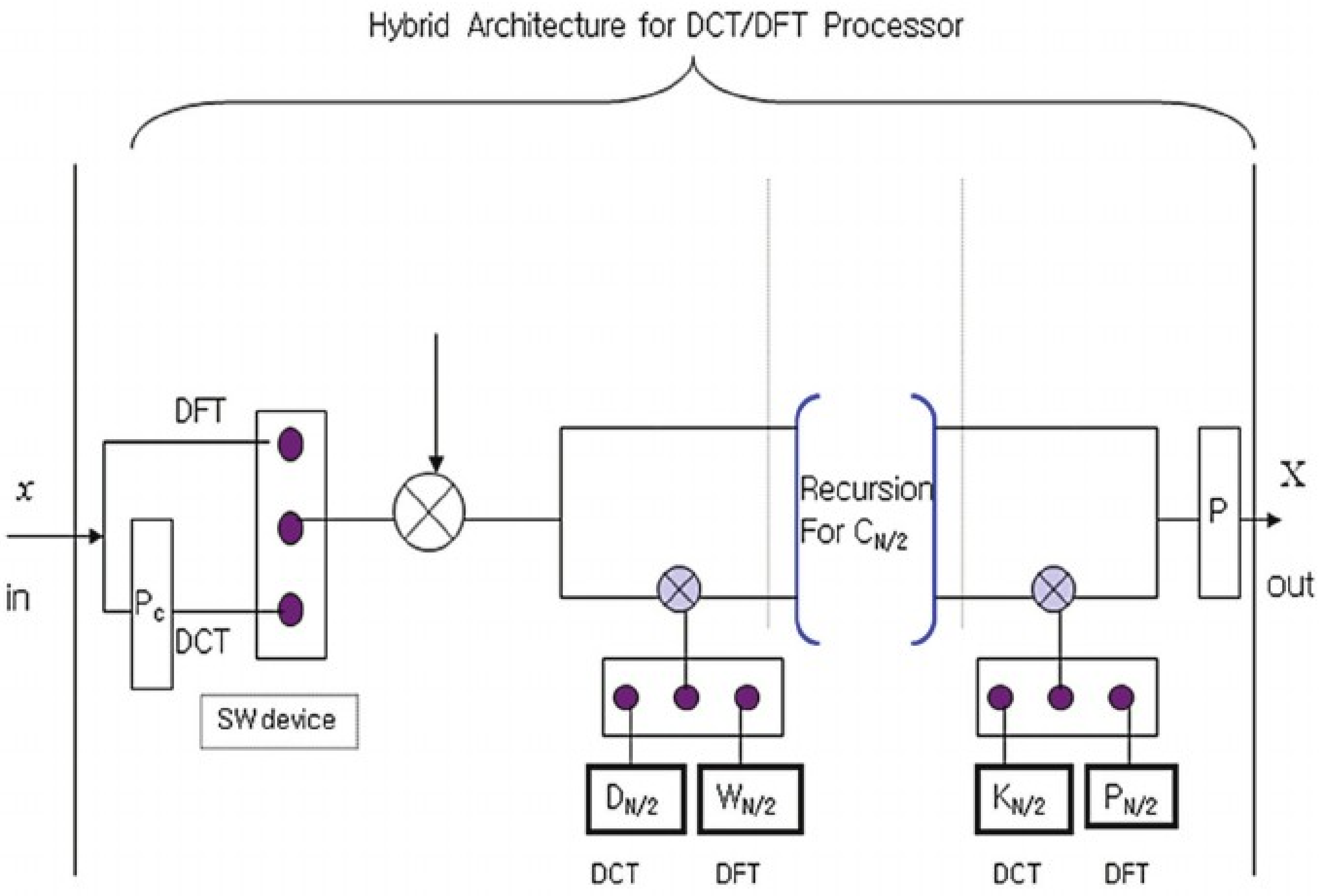


*	R	ARG1	R	ARG2	Dest. Of Result

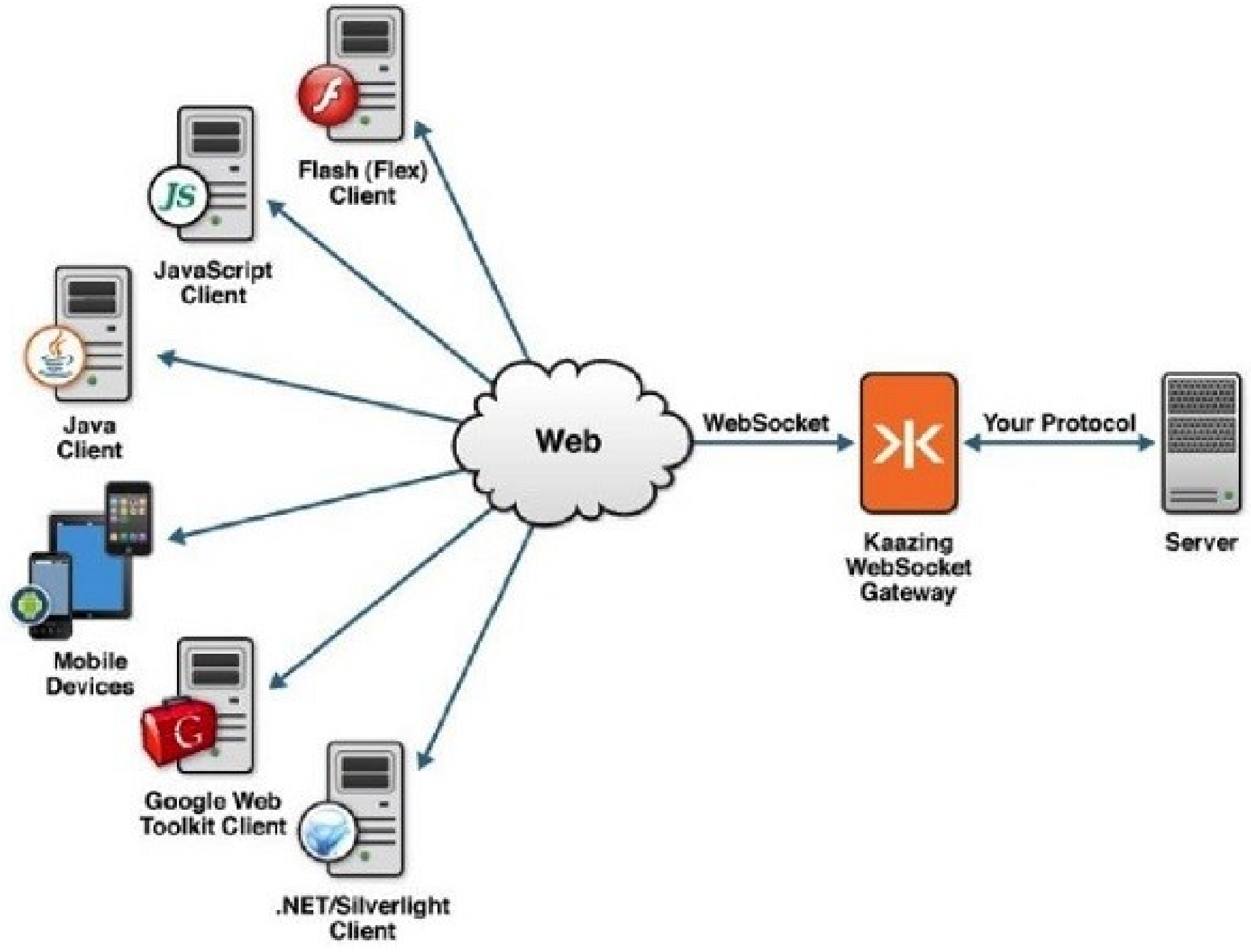
Explicit Token Store Machines

- A greatly simplified approach to data-flow execution, called the explicit token store (ETS) architecture, and its current realization in Monsoon are presented.
- The essence of dynamic data-flow execution is captured by a simple transition on state bits associated with storage local to a processor .
- The Explicit Token Store (ETS) architecture is an unusually simple model of dynamic dataflow execution that is realized in Monsoon, a large-scale dataflow multiprocessor.
- A Monsoon processor prototype is operational at the MIT Laboratory for Computer Science, running large programs compiled from the dataflow language Id.

Hybrid Architecture



Unified Architecture


Flash (Flex) Client
JavaScript Client
Java Client
Mobile Devices
Google Web Toolkit Client
.NET/Silverlight Client
Web
WebSocket
Kaazing WebSocket Gateway
Your Protocol
Server