Theory 3 Tutorial 1 Total - 100.

INDEX

NAME: Ruby Shilpakar. STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign/ Remarks
		Advanced Computer Architecture.	Assignment - 2 hp 7 Assgns.	
1.		CL : 1 long question.		
1.	Concept	1.1 Basic computational model		6hrs
		1.2 key concept relating to CM.		
		1.3 Concept of comp. Arch. with interpretations & descriptions.		
2.	Intro. to parallel processing	2.1 Introduction, architectural classification schemes.		6hrs
		2.2 Evolution of parallel processors, current & future trends towards P.P		
		2.3 principles of pipelining & array processing		
		2.4 Scalar & vector pipelines		
3.	Vector & pipelined processor	3.1 classification of pipelined processors, performance evaluation factors		
		3.2 Vector processing concepts, pipelined vector processors, Cray type vector processor - design example.		
		3.3 Array processors, an eg of data routing in array processor.		6hrs
		3.4 systolic arrays & their applications		
4.	Different parallel processing architecture	4.1 Introduction to Associative memory processors		6hrs
		4.2 Multithreaded arch - principle of multithreading, latency hiding		
		4.3 Scalable coherent multiprocessor model with distributed shared		
5.	Distributed Memory Architecture	5.1 loosely coupled & tightly coupled architecture		6hrs
		5.2 Cluster computing as an application of loosely coupled architecture. Eg; CM* and Hadup.		
6.	programmability Issues	6.1. Types & levels of parallelism		5 marks.
		6.2 OS for parallel processing, models of parallel OS - ~~Master slave~~ slave conf, separate supervise config, floating supervisor control		3hrs
7.	program & network properties (1 Topic)	7.1 Conditions of parallelism		
		7.1.1 Data & resource dependencies		
		7.1.2 Data dependency analysis - Bernstein's conditions		1.8
		7.1.3. Hardware & SW parallelism.		
		7.1.4. The role of compilers		S.M

S. No.	Date	Title	Page No.	Teacher's Sign/ Remarks
		7.2 Program partitioning & scheduling		5 hrs.
		7.2.1 Grain sizes & latency		15-20 mar
		7.2.2 Grain packing & scheduling		
		7.2.3 Static multiprocessor scheduling		
		7.3. System interconnect Architectures		
		7.3.1 NIW properties & routing		
		7.3.2. Static connection networks.		
		7.3.3. Dynamic connection networks		
				8hrs.
8.		Parallel models, language & compilers.		20 marks

items of comput

→ Interpretation, basic concept, execution

Date 02/2074
Page No. Wednesday.

Chapter 1:
Concepts [on Computation : Model]

Computation:

2015 -Computation is defined as information processing that can be represented mathematically

-Model can be termed as foundation or paradigm of the given aspect or system. Model expresses the level of abstraction of the given aspect or system.

2015 Computational model can be expressed with two aspect, they are:

1) Computer architecture
2) Computer language.

2015 -Computational model can be defined as the foundation or paradigm which can express information processing techniques and methods, its affects with information aquisition and processing which can be expressed mathematically. So, for this purpose the computational model can be interpreted into three concept they are:

a. Basic items of computation.
b. Problem Description.
c. Execution Model.

Page No.

Let us consider an example for the computation of Tower of Hannoi problem where its solution steps or the information processing can be represented in mathematical form in $2^n - 1$. where n is the number of disk.

Relationship bet= Computational Model, Computer language & Computer Architecture

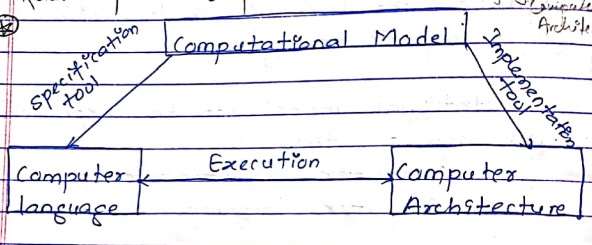


Fig: Relation between computation model, computer architecture and computer language.

Computer language may be procedural, declarative or object oriented which helps to execute the computation model with the use of specification tool like LIPS, JAVA etc.

Computer Architecture helps to implement the computational model for its execution by the use of implementation tool (Platform like Windows, Linux etc.)

Page No. Date / /

Concept / Interpretation of Computational Model: (Abstraction)

1. Basic Items of computation
Data, object, arguments and functions, element of sets and the predicates can be basic item of computation.

2. Problem Description Model
It interprets the computational model in two manner they are:
style
Method.

Problem Description style : Procedural
- Algorithm for solving the problem is stated.

Eg:
```
int factorial (int n)
{
  int facto = 1;
  if (n>0)
    for(int i = 1; i<=n; i++)
      facto = facto * i;
    return facto;
}
```

* Declarative
- All facts and relationship relevant to the given problem is stated.
- Using function in a model called applicative.
- Using predicate in model called predicate

logic based.

Eg:
fact(0) = 1;
fact(n>0) = n * fact(n-1);

Problem Description ~~Method~~ Method:
- Procedural
How a solution of the given problem has to be described.
Eg: Sequence of instruction.

- Declarative
How the problem itself has to be described.
Eg: set of function.

3. Execution Model
Execution model interprets the computational mode as:

* Interpretation of how to perform the computation which is related to problem description method.

* Execution semantics
It is the rule that describes how a single execution step is to be performed.

Execution Semantic consist of:
i) State Transition Semantic
- turing model
- Von-Neuman model
- Object based model.

ii) Dataflow Semantic
- Dataflow Model.

iii) Reduction Semantics
- Applicative Model.

iv) SLD - resolution Semantic
- Predicate logic-based model.

Control of the execution Sequence (Execution Control)
It assist in ordering of execution sequence and control the sequence of execution with three aspect they are:
i) Control Driven:
It assumes that there exist a program consisting of sequence of instruction so that execution sequence is then implicity given by the order of the instruction & explicit control instruction to change the state.

ii) Data Driven:
In this, an operation is activated as soon

as all the needed input data is available. eg. Dataflow Model

(ii) Demand Driven

In this an operation is activated only when execution is needed to achieve the final result.

eg. Applicative Model

1.1

Computational Model

Aspects	Turing	Von-Neuman	Object-based	Dataflow	Applicative	Predicate logic based
Basic item of computation	Elements of the tape, symbol sets, Mapping definition	Data assigned to named entities (variables) operational performed on data	Object which can be manipulated by set of msg, msg sent to manipulate object	Variables, operation performed on data	Arguments, Application of function to their arguments.	Elements of sets, Predicate declare to them.
Problem description model — Style	Procedural	Procedural	Procedural	Procedural	Declarative	Declarative
Description	I/p data → Transition function (δ), initial state q_0	Input data → Sequence of instruction → output data	Initial state of object → Sequence of msg sent to create object → final state of object	Input data → output data	Arguments of the fxn to be evaluated → set of function definition → Evaluated fxn	Goal statement → Set of definite clauses of expressing rules & facts → Logic Inference
Interpretation	starting with the initial state sequence of subsequent state will be performed until final state is reached.	The given sequence of instruction will be executed using input data	The given sequence of msg will be executed.	I/p data flows via the dataflow graph & as a result o/p data is obtained	The fxn with the given argument is subsequently rewritten by substituting the fxn definition referenced and ⊕	The goal statement is subsequently rewritten using resolution and logical inference is further done more unification and
Execution model — Execution semantic	State transition semantic	State transmission semantic	State transmission semantic	Dataflow semantic	Reduction Semantic	SLD-resolution semantic
Control of execution sequence	Control Driven	Control Driven	Control Driven	Data Driven	Demand Driven	Defined by both the goal processing rule & search rule
Concurrency	Sequential	Sequential	Sequential	Parallel	Parallel	Parallel
Language class	Type 0	Procedural	Object oriented	Single Assignment	Functional	Logic Programming
Architecture class	—	Von-Neumann	Object oriented	Dataflow	Reduction	Unknown

⊕ performing the possible reduction untill no more substitution or reduction are feasible.

1.2 Key concepts relating to computational Models:

1. Granularity:
One of the vital decisions relating to a computational model concerns the types of items that the computation should be performed on and the kinds of operation allowed on them. Possible alternatives are data, objects, sets and so on, along with the declared operations. These items and operations may, however be of quite different complexity. In order to reflect the complexity of the items of computation, it makes sense to introduce the concept of granularity. Then items of computation can be informally classified as fine-grained or coarse-grained according to their complexity: in some cases middle-grained items of computation can also be distinguished. Accordingly, as far as Von-Neumen architectures are concerned, RISC, conventional CISC, HLL and application-oriented architectures may be distinguished as subclasses with increasing granularity.

The concept of granularity was introduced in connection with parallel architectures as well. In this case the granularity refers to the size of the computations that can be executed in parallel without any synchronization or communication. Usually, the parallelism of individual operations is designated as fine-grain parallelism, while parallelism at the thread or process level is called coarse-grained parallelism. Clearly, the computation / communication ratio is better in coarse given grain systems. On other hand the exploitable parallelism is usually much less in coarse-grain systems than in fine-grain parallel systems. It is useful to introduce the concept of granularity at a higher abstraction level as well, that is in connection with computation models. In sequential environments the granularity is interpreted as the complexity of the granularity is interpreted as the complexity of the items of computation, whereas in parallel environment as the size of parallel computations. This interpretation is vague and imprecise but concerning granularity it yields a unified framework for both architectures and language.

Architecture class.

Granularity Reduction	Von Neumann	Dataflow
Low	RISC	Fine grain
grain		
(Medium grain)	conventional CISC	(Medium grain)
	HLL	
high	Application oriented	Coarse grain

The interpretation of granularity for different from architecture class.

2. Typing:

Typing is another concept which is worth introducing at a higher level as it is often used in connection with programming languages. If the concept of typing is introduced at the abstraction of level of the computational model, typing of languages and tagging of architectures are closely associated. In typed languages, there exists a concept of data type and the language system (compiler and interpreter) may check the consistency of the types used in expressions function invocations and soon. If the language is strongly typed, a type mismatch will cause an error. In weakly typed languages a type mismatched is allowed under given circumstances namely if the types involved are consistent.

LISP and FP are examples of untyped language. Pascal, Mirandra and HOPE are strongly typed languages whereas the single assignment languages sisal is weakly typed. Typed architectures are commonly called Tagged. They provide a mechanisms for typing the data being stored or processed by extending the data word by a Tag. The tags (usually 3-5 bit long) contain type-identification.

Granularity	Language class
Low	conventional assembly language
High	conventional high-level language.

The interpretation of granularity for programming language

programming language.

p.T.O.

Date 23/02/2074
Page No. Tuesday.

2016.

1.3 Concept of Computer Architecture with interpretation & Description.

→ Computer Architecture was coined by IBM system and interpretated as structure of a computer that a machine language programmer must understand to write a correct program for machine, where interpretation comprises the defination of register and memory as well as of the instruction set, instruction format, addressing modes and the actual coding of the instruction excluding implementation and realization. By implementation, actual hardware structure can be understood and by reduction, logic technology, packing and interconnection can be understand.

→ The concept of computer, architecture can be described by introducing a hierarchial multi-level description where four levels are identified that can be used for describing a computer, they are:

1) Electronic circuit level
2) Logic Design level
3) Programming level
4) PMS level.

→ Programming level refers to concept or architecture. Processor - Memory - switch level is a

top level description of a computer system based on the specification of the basic units like processor, memory, etc. as well as their interconnection.

→ Computer Architecture concept a "two dimensional" interpretation. They are level of Abstraction and other is scope of

→ interest. -Although interpretation of architecture concept is quite useful for a broad range of computer, it assumes the use of Von-Neuman Model of com-

→ putation and for generalize concept of computer we third dimension is included.

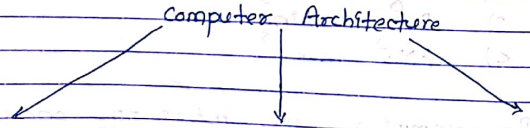
architecture, we must define

Recent interpretation of computer Architecture.

→ The concept of computer Architecture is interpreted at number of levels of increasing abstraction, where at

→ each level, architecture is described by Under lying computational model, func-tional specification as well as actual

→ implementation. So, interpretation can cover theree aspect and they are:



In past days, Computer Architecture was inherently interpreted as a Von-Neumen Architecture but in present days more CM has been developed, so concept of CA should include the specification of underlying computational model.

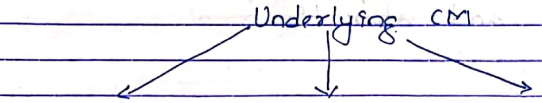


Fig: Most papular CM related to Architecture.

Concerning the level of consideration, there are mainly three level of interest in increasing degree of Abstraction.

- Micromachine level
- Process level
- Computer system level.

Architecture can be used at each level of consideration with two distinct scope of interest they are:

- Abstract Architecture
- Exo Architecture / Logical Architecture / External Architecture.
- Interested in functional specification of a computer.

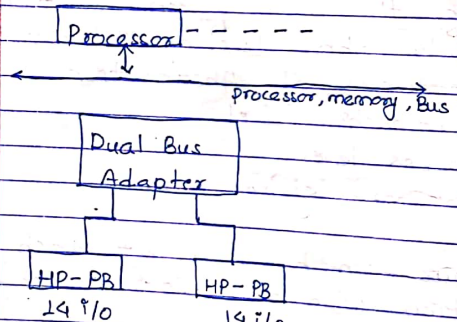
- Reflects black box view.

Concrete Architecture.
Physical Architecture / Endo Architecture/ Structural Description.

- Reflects description of internal structure and operation.

Computer Architecture at different level of Abstraction.

1. Concrete Architecture of computer system.



* HP-PB refers HP-Precision Bus.

2. Abstract Architecture of Processor.

Human → computer
implementation
(internal structure and operation)

Fig:- Interpretation of processor level Abstract Architecture.

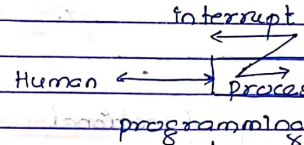


Fig:- Interpretation of process-level concrete Architecture.

3. Concrete Architecture of processor.

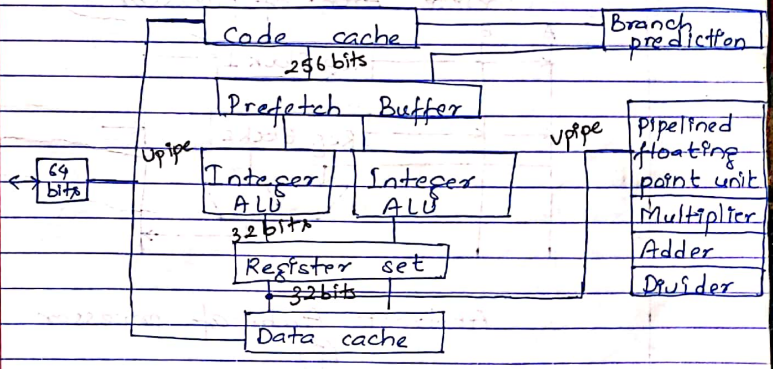
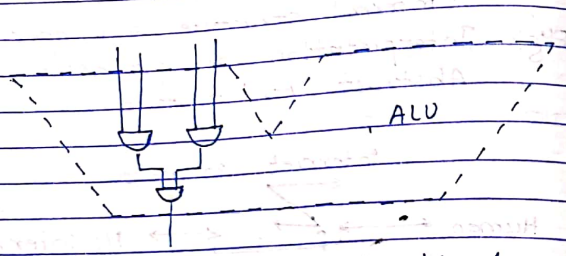
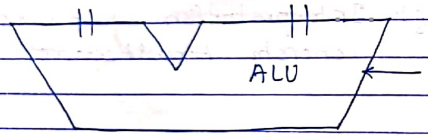


Fig:- Logical Design of processor.

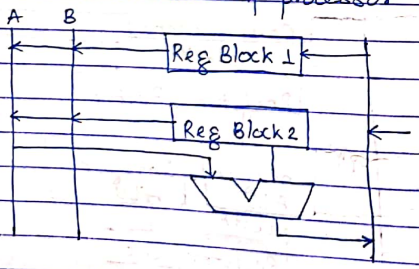
Eg:- Concrete Architecture of functional blocks.



Abstract Arch. of functional Block.



Concrete Arch. of processor

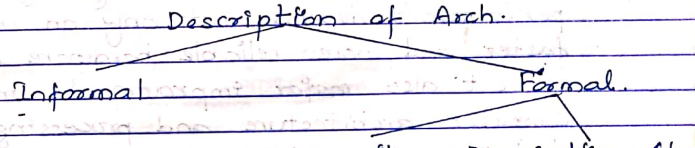


Abstract Arch. of processor

(P)

Description of Computer

- CA is formally described by using an Architecture Description language (ADL)
- and it is also termed as HDLs (Hardware Description language) or CHDLs (computer Hardware Description Languages).
- Register Transfer Language (RTLs) designates a particular type of ADL which is confined to processor level - HDLs are considered as low-level and ADLs as higher level description language.



Description of by ADLs

(HDL, CHDL & RTL)

A fixed set of design decision is assumed.

Description of by design space using DS trees

All considered design aspects and available choice for each of the aspect can be used.

Chapter 2:

Introduction to Parallel Processing.

2.1

High performance computers are increasing in demand in the areas of structural analysis, weather forecasting and many other scientific and engineering application, where many of these challenges to this advance civilization cannot be constructed within reasonable time period without super power computer. So, achieving high performance depends not only on using faster and more reliable hardware devices but also major improvements in computer architecture and processing technique. So, parallel processing is accounted, which are used by parallel computers.

Parallel computers are those that emphasize the parallel processing between the operations in some way and they can be characterised based on the data and instruction stream forming various types of computer organization. They can also be classified based on computer structure. eg: multiple processor having separator memory or share global memory.

Data stream ——
Instruction stream - - - -

Architectural Classification Scheme.

Flynn's classification
Feng's classification.
Handler's classification.

Flynn's Classification:
[Proposed by Michael Flynn]

Flynn introduced the concept of instruction stream and data stream for categorizing of computer but not all are parallel computers. the concept is use to grasp the concept of parallel computer which is necessary to understand the classification.

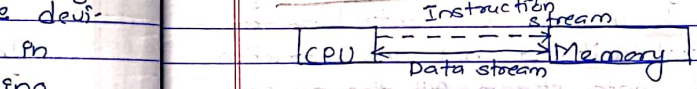


Fig:- IS & DS

Flynn's classification is based on multiplicity of instruction stream and data stream observed by the CPU during program execution. The computer organization can be classified as

Single Instruction & single Data Stream (SISD)
In this organization, sequential execution of instruction is performed by one CPU containing a single Processing element (P.E) ie. ALU under one control unit.

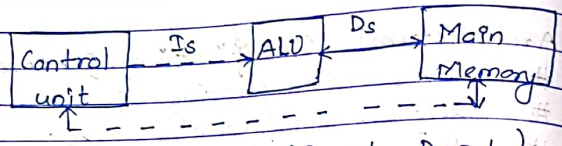


Fig: SISD ($I_S = 1$, $D_S = 1$)

SISD machines are conventional serial computer with the processors, only one stream of instruction & one stream of data. Eg PCs, Cray-1, CDC 6600, CDC 7600 etc.

SIMD (Single instruction & Multiple Data stream)

In this organization, multiple processing element work under the control of a single control unit. It has one instruction and multiple data stream. All the processing element of this organization receive the same instruction broadcast from the CPU. Main memory can also be divided into Modules for generating multiple data stream acting as a distributed memory.

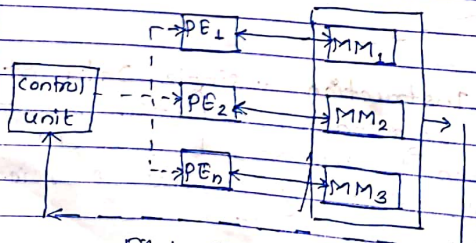


Fig: SIMD ($I_S = 1$, $D_S > 1$)

All the PEs, simultaneously execute the same instruction and are said to be 'lock-stepped' together, where each processor takes the data from its own memory and hence it has distinct data stream and must be allowed to complete its instruction before next instruction is taken for execution. Eg:- ILLIAC-IV, PEPE, BSP, DAP, IBM 9000, Fujitsu etc.

Multiple Instruction Single Data stream.

In this organization, multiple processing elements are organized under the control of Multiple control units, where each control unit handles one instruction stream and processed through its corresponding processing element. But each processing element is processing only a single data stream at a time. Used for handling multiple instruction and single data stream, multiple control units and multiple PEs are organized. All PEs interact with the common shared memory for the organization of single Data stream. Only known example of MISD is C.mmp built by Carnegie-Mellon University and not popular in commercial machine.

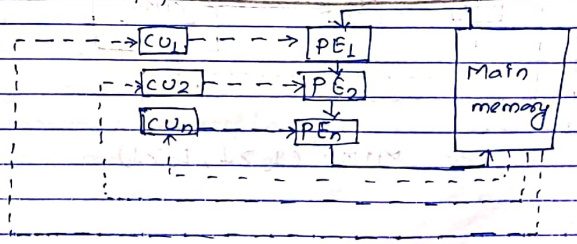
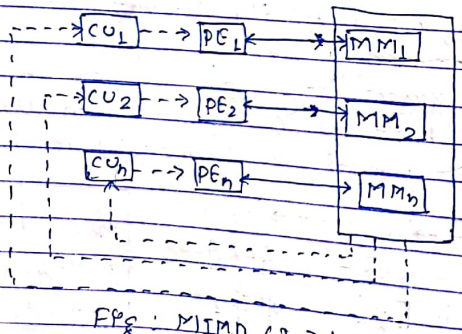


Fig: MISD ($I_S > 1$, $D_S = 1$)

Multiple Instruction Multiple Data stream (MIMD)

In this organization, multiple PEs and multiple CUs are organized as in MISD, where multiple instruction stream operates on multiple data stream and for handling multiple instruction stream, multiple CUs and multiple PEs are organized to handle multiple data stream from the main memory. The processors work on their own data with their own instruction where task executed by different process can start or finish at different time & runs asynchronously. It actually recognises the parallel computers and microprocessor system fall under this. Eg. C.mmp, Cray-2, Cray-XMP, Univac 1100/80 etc.



Eg: MIMD (x≥1, y≥1)

Feng's classification

Tse-yun Feng suggested the use of degree of parallelism to classify various computer architecture where maximum number of binary bits that can processed within a unit time by a computer system is maximum parallelism degree P. A bit slice is a string of bits one form each of the words at the same vertical position. The organization can be

- Word Serial & Bit Serial (WSBS)
  It is called bit parallel processing because one bit is processed at a time.
- Word Parallel & Bit serial (WPBS)
  It is called bit slice processing at a time.
- Word serial & Bit Parallel (WSBP)
  It is found in most existing computer & has been called as word slice processing because one word of n bit processed at a time.
- Word Parallel & Bit Parallel (WPBP)
  It is called as fully parallel processing in which an array of n×m bits is processed at a time.

Handler's classification

Wolf gang Handler has proposed a classification scheme for identifying the parallelism

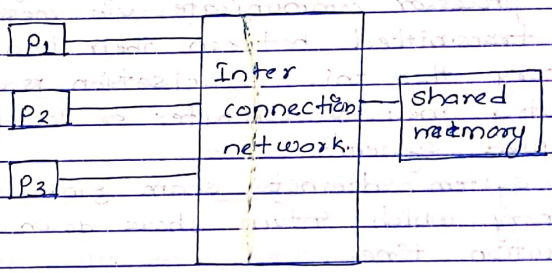
degree and pipelining degree built into the hardware structure of computer system where it consider three sub system level [Processor control unit, ALU, Bit level circuit]. Each PCU corresponds to one processor or a CPU, ALU is equivalent to PE and BLC corresponds to combinational logic circuitary needed to perform one bit operation in ALU.

Flynn's classification only discuss the behavioural concept and doesnot take computer's structure. Parallel computer also can be classified on the basis of structure. A parallel computer can be characterized as a set of multiple processors and shared memory and memory modules communicating via an interconnection network.

Structure of Parallel computer
- Tightly coupled system
- Loosely coupled system.

When multiprocessors communicate via the global shared memory modules then this organisation is shared memory computer or Tightly coupled system.

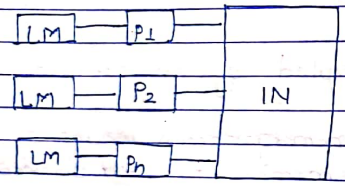
They are used for high speed real time processing.



Tightly coupled system can be classified as:

- Uniform memory access model.
  - main memory is ~~used~~ shared uniformly by all processor.
  - used for time sharing app.
- Non UMA.
  - Local memory, can be connected with every processor.
  - collection of local memory form global memory.
- Cache - only memory architecture. (COMA).
  - used cache memory with every processor for reducing execution time.
  - NUMA with cache instead of local memory.

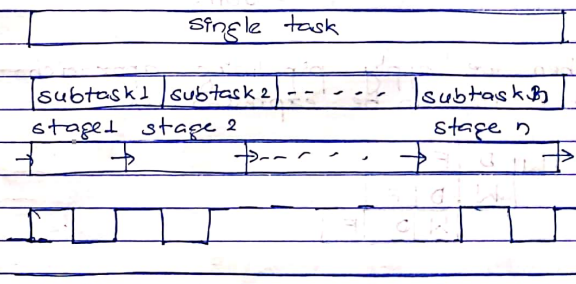
When every processor in multiprocessor system, has its own local memory & the processor communicate via messages transmitted between their local memory then this organisation is distributed memory computer or locally coupled system.
The system doesnot share global memory which inturn shows down execution time.



2074102130

2016
2.3
Principles of Pipelining:
Pipelining is an implementation technique where multiple instruction are overlapped in execution. Computer pipeline is divided into stages where each stages completes a part of an instruction in parallel. Pipelining is the key technique used to make fast CPU.

Principles:
- Work to be done is broken down into smaller steps.
- Pipeline stages associated with each subtask has equal amount of time as available in each stages for performing required subtask.
- All pipeline stages operates like an assembly line i.e. each stages receives their input typically from previous stages and delivering their output to next stage.
- Pipeline operates synchronously.



Let us consider an example of laundry system where it consist of n laundry. where it takes 40 minutes to wash, 30 minutes to dry and 20 minutes to fold for a single laundry. When a conventional approach is followed, then it can be expressed as:

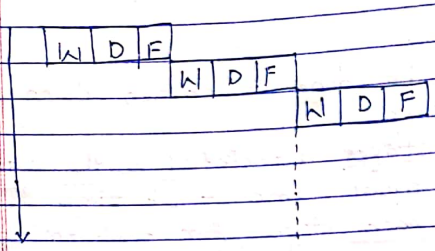
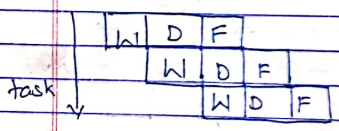


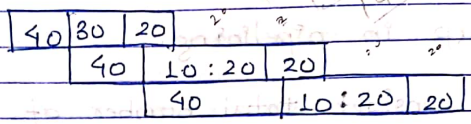
Fig: Conventional Approach in laundry system.

So, total time (T) = $n * (t_W + t_D + t_F)$
= $n * (40 + 30 + 20)$
= $90n$.

When we apply pipelining approach then it can be expressed as:



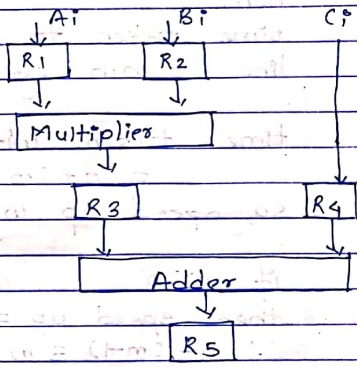
Approximate time can be calculated as (T), assuming n=4.
So,
$(40 + 30 + 20) + 40 + 40 + 40$
= 210



Now, considering next example $A^i \times B^i + c^i$ for $i = 1 \ldots 7$

sub operation performed in each segments are:

$R_1 \leftarrow A^i$
$R_2 \leftarrow B^i$
$R_3 \leftarrow R_1 * R_2$
$R_4 \leftarrow c^i$
$R_5 \leftarrow R_3 + R_4$



Contents of Register are in pipeline which can be expressed as:

Clock pulse	segment 1		segment 2		segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	c_1	—	—
2	A_2	B_2	$A_1 * B_1$	c_1	—
3	A_3	B_3	$A_2 * B_2$	c_2	$A_1 * B_1 + c_1$
4	A_4	B_4	$A_3 * B_3$	c_3	$A_2 * B_2 + c_2$
⋮	⋮	⋮			⋮

Page No.

Speed up in pipelining.

Let us consider total number of instruction executed is m. let number of clock cycle per instruction (with n stages taking 1 clock cycle for each stage) is n.

Then,
time taken to execute without pipeline = nm cycle.

time taken with pipeline (ideal)
= n+(m-1) cycles.
so, speed up with pipelining = $\frac{nm}{n+(m-1)}$

if m>>n
then, speed up = x.
∵ n+(m-1) = m

Eg:

Segment	clock cycle. 1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

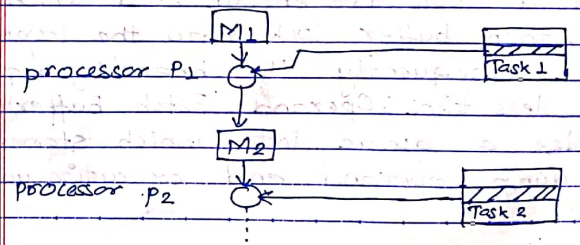
Page No.
Date / /

cycle time (t_p) = 20 ns
No. of segment (k) = 4
No. of task (n) = 100
Pipeline system will take
$(k+n-1) * t_p$
$= (4+100-1) * 20$
$= 2060$ ns

Assuming $t^n = kt_p = 4 * 20 = 80$ ns
A non-pipelining system requires nkt_p
$= 100 \times 80 = 8000$ ns.

The speedup ratio = 8000 / 2060.
= 3.88

Processor Pipelining
It refers to pipelining of data stream by a cascade of processors each of which processes a specific task. The data stream passes the first processor with results stored in a memory block which is also accessible by second processor and it passes the refined results to third and soon.



Compare bet[illegible]

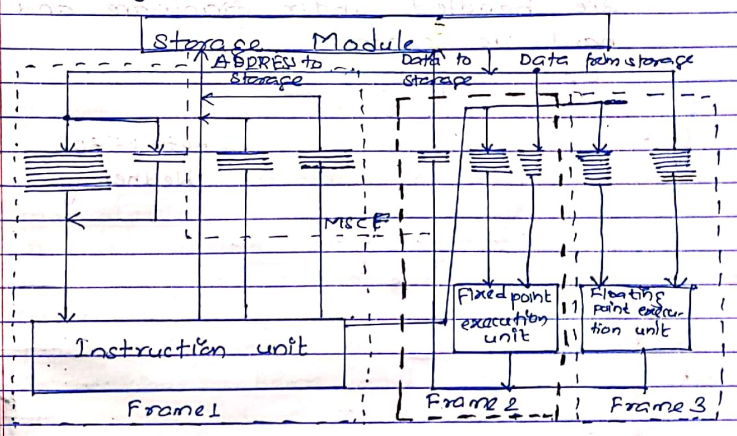
2.4 Scalar and Vector Pipelines:

On the basis of instruction or data types, pipeline processor can be classified as scalar and vector pipeline.

Scalar pipelines processes a sequence of scalar operands under the control of 'do' loop. Instruction in a small do loop are often, refetched into the instruction buffer. The required scalar operand are moved into a data cache for continuously supply to pipeline with operands.
Eg: IBM system /360 Model 91.

In IBM system, buffering plays vital role and buffering can be of instruction fetch buffering and operand fetch buffering. Instruction fetch buffering provides the capacity to hold program loops of meaningful size.

When a loop is encountered which fits, then buffer lock onto the loops and subsequently the branching requires less time. Operand fetch Buffering provides a queue into which storage can dump operand and execution units can fetch operands which can improve operand fetching for storage, to ~~store~~ register and storage to storage instruction type.



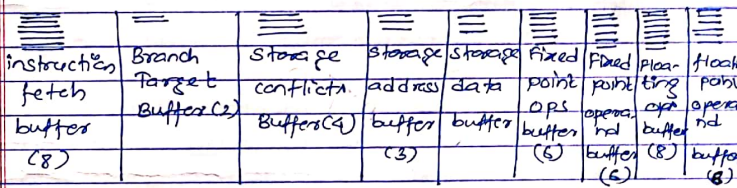


Fig:- IBM System Architecture

Vector pipeline are specially designed to handle vector instruction over vector operands. Vector operands in vector pipeline are handled under firmware & hardware control. Eg:- Cray 1.
MSCE: Memory storage control element.

Vector pipeline are specially designed to handle vector instruction over vector operands.
Vector operands in vector pipeline are handled under firmware and hardware control.
Eg: Cray 1.

2074/02/30.
Wednesday

P.T.O

Date 30/02/2074
Page No. Wednesday.

Chapter 3:
Vector and Pipeline Processor.

A vector processor or array processor is a cpu that implements an instruction set containing instruction that operates on one-dimensional array of data called vectors. They can greatly improve the performance.

Pipelined processor is a CPU that implements pipelining approach i.e. the stages of the instruction are over-lapped in execution. Pipelining is accounted in vector processor and they are termed as vector pipelines. Pipeline processor can be classified in various scheme and two vital scheme are:

1) Handler's classification.
2) Li and Rammurthy's classification.

Handler's classification.
On the basis of level of processing the pipelined processor can be classified as:
- Arithmetic Pipeline
- Instruction Pipeline
- Processor Pipeline

Arithmetic Pipeling.
In this, the CPU of computer can be segmented for pipeline operation in various data formats.

formats: Eg: Star 100.

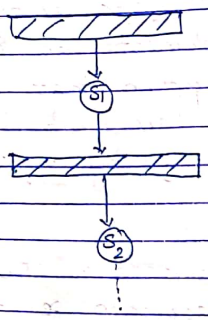


Fig: Arithmetic Pipelining.

Star 100 has two pipelines where arithmetic operations are performed, they are:

a) Floating point Adder & Multiplier
b) Multifunctional,

which are used for all scalar instruction with floating point adder, multiplier and divider. In this, both pipelines are of 64 bits and can be applied into four 32-bits at the cost of precision.

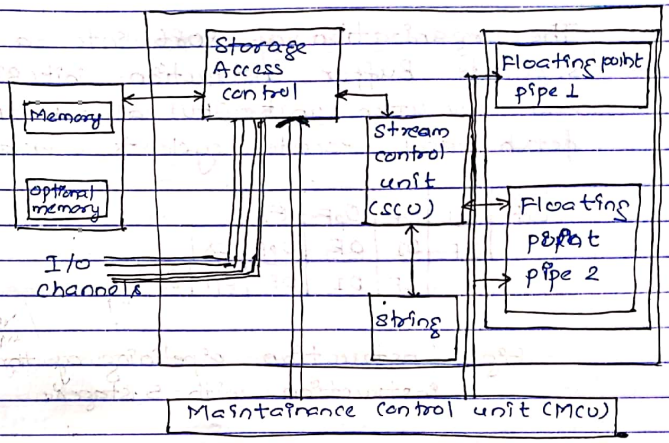


Fig:- Block Diagram of Star 100.

Instruction Pipelining.

In this approach, execution of a stream of instruction can be pipelined by overlapping the execution of current instruction with fetch, decode and operand fetch of subsequent instruction and this process is called instruction look-ahead. Eg:- 8086

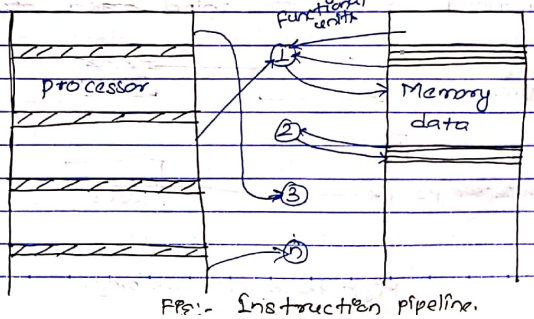


Fig:- Instruction pipelining.

The organization of 8086 into a separate Buffer Instruction Unit (BIU) and Execution unit (EU) allows to fetch and execute cycle to overlap.

IF	DI	OF	OpF	EX	
	IF	DI	OF	OpF	EX
		IF	DI	OF	OpF	EX

Fig: Instruction pipelining of three instruction with 5 stages.

Processor pipelining / pipelining. (chapter 2 copy)

It refers to the pipelining of data stream by cascade of processor, each of which processes a specific task.

The data stream passes the first processor with result stored in memory block which is also accessible by the second processor and it passes the refined result to third and so on.

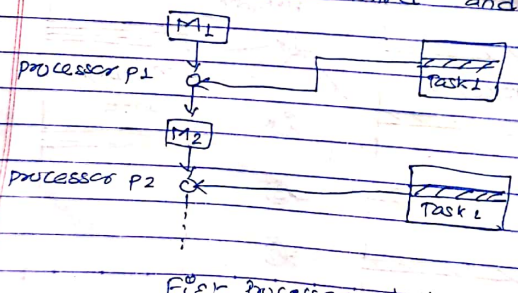


Fig: Processor pipelining.

Li and Ramamurthy is classification.

On the basis of pipeline configuration and control strategies, Li and Ramamurthy classified pipeline under three scheme.

1) Unifunctional and multifunctional pipeline.
2) Scalar and vector pipelines
3) Static and dynamic pipelines.

Unifunctional and multifunctional pipeline.

- Unifunctional pipeline has a pipeline unit with fixed and dedicated function.
  Eg: CRAY 1 (super-computer - 1976)

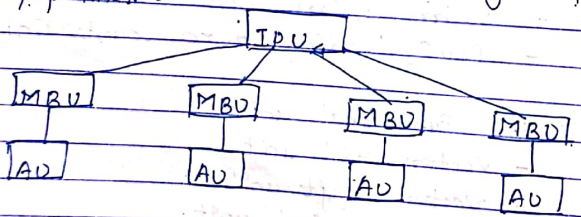
CRAY 1 has 12 unifunctional pipelines which are describe in four groups:

1. Address functional units
   - Address Add units.
   - Address Multiply units.
2. Scalar functional units
   - Scalar Add units
   - Scalar shift units
   - Scalar logical units
   - Pop / leading zero count unit
3. Vector Functional units. (array)
   - Vector Add unit
   - Vector shift unit
   - Vector logical unit.

4. Floating point functional unit.
   - Floating point Add unit.
   - Floating point Multiply unit.
   - Reciprocal Approximation unit.

Multifunctional pipelines performs different function either at different time or same time by interconnecting different subset and of stages in pipeline.
Eg: 4X-TI-ASC (supercomputer -1973)

4X-TI-ASC has four multifunctional pipeline processor each of which is reconfigurable for a variety of arithmetic or logic operations at different times. It is a four central processor which is consist of nine units. It has one instruction processing unit, four memory buffer unit and four arithmetic unit. Hence, it provides four parallel execution pipeline under IPU which helps to execute any mixture of scalar and vector instruction simultaneously on four pipelines.



Static and Dynamic Pipelines.

Static pipelines assume only one functional configuration at a time so it can be either unifunctional or multifunctional. They are preferred when instruction of same type are to be executed continuously. Normally a unifunctional pipe must be static.

Dynamic pipeline permits several functional configuration to exist simultaneously so it must be multifunctional. Dynamic configuration requires more elaborate control and sequencing mechanism than static pipeline.

Scalar and vector pipeline. Refer chap 2.

Date 01/03/2074
Page No. Thursday

Assignment.

Explain the factors & give example.

Performance Evaluation Factors (of pipeline)

Speed up $= \frac{nk}{k+(n-1)}$

Efficiency $= \eta = \frac{n}{k+n-1}$

Throughput $= \frac{n}{(k+n-1)T}$

$= \eta * \frac{1}{T}$

$= \eta * f$

Speed up is defined as ratio of time taken for given computation by a non-pipeline functional unit, to time taken for same computation by a pipeline functional unit. Let us assume a function with k stages of equal complexity which takes the same amount of time T. In non-pipelined function it will take kT time for one input. let there are n inputs. so,

Speed up $= \frac{nkT}{(k+n-1)T}$

$\therefore$ Speed up $= \frac{nk}{k+n-1}$

Date / /
Page No.

Eg: If a pipeline has 4 stages and 5 inputs its speed up factor,

$= \frac{5*4}{4+5-1}$

$= \frac{20}{8}$

$= 2.5$

The maximum value of speed up is

$\lim_{n\to\infty}$ [speed up]

$= \lim_{n\to\infty} \frac{nk}{k+n-1}$

$= k$ //

Efficiency is an indicator of how efficiently the resources of pipelines are used. If a stage is available during a clock period, then its availability becomes the unit of resources. Efficiency can be defined as ratio of Number of stage time units actually used during computation to total number of stage time units available during that computation.

No. of stages time unit $= nk$ where n is number of inputs having k stages.

Total number of stages time unit available: = k(k+n-1)

i.e. product of number of stages in pipeline (k) and number of clock periods taken for computation (k+n-1)

so,

$$\eta = \frac{nk}{k(k+n-1)}$$

$$= \frac{n}{k+n-1}$$

for maximum efficiency

$$\lim_{n\to\infty} \eta$$

$$= \lim_{n\to\infty} \frac{n}{k+n-1}$$

$$= 1 //$$

Note that η is minimum of n = 1.

Eg:-

n = 5

k = 4

so,

$$\eta = \frac{5}{4+5-1}$$

$$= \frac{5}{8}$$

$$= 0.625$$

$\eta = 62.5\%$

minimum- $\eta = \frac{1}{4+1-1}$

$$= \frac{1}{4}$$

$$= 0.25$$

$\therefore \eta_{min} = 25\%$

Throughput is the average number of results completed per unit time.

for n inputs, a k-staged pipeline takes [k+n-1]T time inputs units.

The throughput $= \frac{n}{(k+n-1)T}$

$$= \left[\frac{n}{k+n-1}\right] * \frac{1}{T}$$

$$= \eta * f$$

$\because f = \frac{1}{T}$

$\therefore$ Throughput = Eff * frequency of machine or processor.

Ch. 8 marks

Consider the execution of a program of 15000 instruction by a pipeline processor with a clock rate of 25 MHz. Assume that the instruction pipeline has 5 stages and that one instruction is issued per clock cycle. The penalties due to branch instruction and out of sequence execution are ignored. Calculate value of PEF for given pipelined processor.

∴ Soln:

$k = 5$

$n = 15000$

$f = 25MHz$

$= 25 \times 10^6 Hz$

now,

$$\text{speed up} = \frac{nk}{k+n-1}$$

$$= \frac{15000 \times 5}{5+15000-1}$$

$$= 4.99$$

Again,

$$\text{Efficiency} = \frac{15000}{5+15000-1}$$

$$= 0.9997 = 99.97\ \%$$

And,

Throughput $= \eta * f$

$= 0.9997 * 25 \times 10^6 Hz$

$= 24.99 \times 10^6 Hz$ (CMOIS)

Vector Processor

Normal computational system are not enough in some special processing requirement like special processing system for expert system where the data processing will be involved on very high amount of data. So, instead of processing high amount of data it can be represented in arrays for processing this data, a new approach is encountered and that is called Vector processing which deals with one dimensional array of data.

Eg:

Consider a program which is adding two areas A and B of length 50.

Machine level program.

Initialize I = 0.

Begin,

Read A[I]

Read B[I]

Store C[I] = A[I] + B[I]

Increment I = I + 1

if I <= 50 go to Begin

Continue.

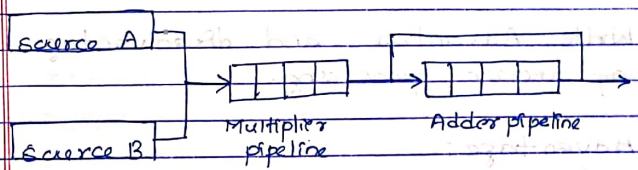
- Here array are being added using loop where the index is started from 0 & loop is continued until the index value reaches 50. The loop contains 5 statements which repeats 50 times. So total cycle of CPU is 250 cycle.

- Now, using the concept of vector processing then unnecessary fetch cycle can be reduced as fetch cycle are used in creation of vectors. The same program can be written as:

C[1:50] = A[1:50] + B[1:50].

When the system is creating a vector like above the original source values are fetched from memory into vector so that data is readily available in vector. When the operation is initiated on the data, the operation will be performed directly on the data and will not wait for the fetch cycle. Hence, total no. of CPU cycle taken is only 50 cycle.

Eg:- Consider a matrix multiplication where row of matrix A is multiplied with column of Matrix B element and result is added



The values of A and B are multiplied and process is expressed as above. Let both matrix are of order n*n. The first n values are taken from source A and is send to multiplier pipeline along with n values from source B and resultant one value is stored in the adder pipeline. Same process is repeated for remaining values and finally result of row to column multiplication are obtained where the next set of values is brought in multiplier pipeline at the same value time of addition operation is being performed. So that operation are performed simultaneously using parallel processing.

- Vector processing naturally requires several data elements for processing so several modules of memory are used where separate data are processed in each processing unit.

Assignment.

Write Advantage and disadvantage of vector processing.

Advantage:

1. High clock rate
2. Fewer misprediction
3. Good memory latency
4. Simultaneous supply of operand.
5. Less memory access and fast processing time.

Disadvantage:

1. Not fast with scalar instruction.
2. Complexity of multiported VRF
3. Difficulty in implementating precise exception.
4. Expensive and code complexity.

Application.

1. Servers, Cinema, Super computing, cluster computing, mainframes etc.

P.T.O

Date 06/03/2020
Page No. Tuesday

Array Processor

In distributed computing several computers works on the same task where their processing power is shared among all the system so that they can perform the task fast. But the distributed computing gives separate resources for each system and each need to be controlled by task initiating system i.e. CPU which is very hard to manage. So, to perform a specific operation involving a large processing, array processor or attached processor can be used instead of distributed computing. SIMD architecture is simplest attached array processor.

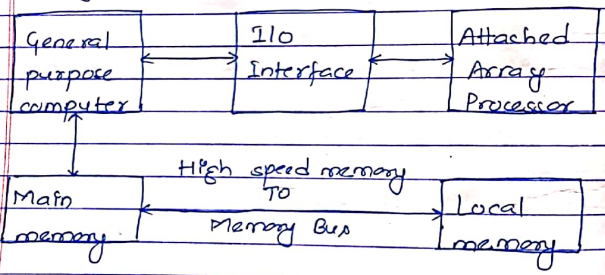


Fig: Attached Array Processor.

The figure expresses that system is attached to a separate processor which will be used for operation of

Date / /
Page No.

specific purpose. Eg: If the array processor is designed for solving floating point arithmetic then, it will only perform that operation.

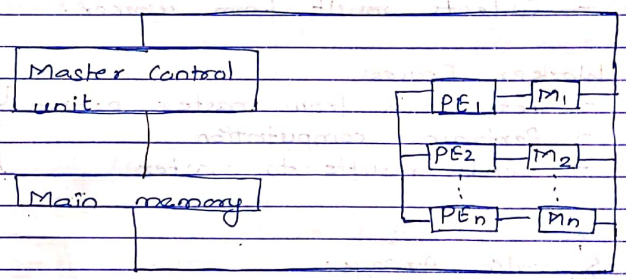


Fig: SIMD Array Processor Technology.

Working Principle (Data Routing)

A master control unit co-ordinate all the process in the array processor, where each processing unit of array processor have local memory unit as in the memory interleving concept on which it performs operation. Main memory consist of original source data and the result that are obtained from array processor.

Master Process
- Holds pool of tasks for worker (slave) processes to do.
- Sends worker a task when requested
- Collects result from worker.

Worker Process
- Gets task from master process
- Performs computation
- Sends results to master

Repeat

5.2.2 Systolic Arrays:

Systolic array is a class of multi-dimensional pipelined array architecture designed for implementing fixed algorithm and are normally designed for performing matrix application.

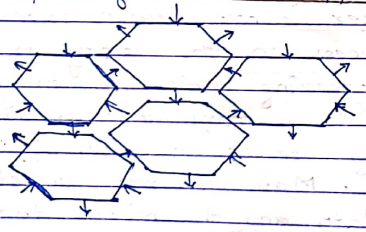
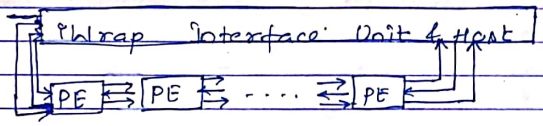


Fig:- Systolic array

They are connected to a small number of nearest neighbour in a mesh-like

topology. Processor performs a sequence of operation on data that flows between them and each processor has same operation and performs operation on data items then pass it to its neighbour. Like SIMD machines, systolic arrays compute in "lock-step" with each processor undertaking alternative phase.

They (static) are pipeline with multi-directional flow of data stream. Eg. Intel iWrap system, it was designed with systolic architecture and becomes popular. A systolic array matches the communication structure of algorithm if it contains fixed interconnection and synchronous operation.



Advantage:
- Faster and scalable

Disadvantage:
- Expensive
- Highly specialized
- Not widely used
- Limited code base of program algorithm.

Application:
- Matrix Arithmetic,
- Signal Processing
- IP
- Language recognition
- Relational DB operation etc. //

Date 07/03/2074
Page No Wednesday

Chapter 4

Different Parallel Processing Architecture.

SIMD computers appears in two basic architectural organization and they are:

Array Processor (using Random Access Memory)
Associative (or content addressable) memory.

Associative Memory (CAM)

It is the special type of memory used in certain very high-speed searching application. It is also called content addressable Memory (CAM), 'Associative storage or associative array. It compares input search data or tag against a table of stored data and return the address of matching data (or in case of associative memory, the matching data) Goodyear STARAN, PEPE were built to implement CAM to design associative computer. In such computer CAM is designed such that user provides a data word and the CAM searches its entire memory to see if that data word is stored anywhere in it. If the data word is found, the CAM return a list of one or more storage address where the

word was found and in some architecture it returns data word or associated piece of data. So, CAM can be considered as hardware embodiment of what in software terms would be called an associative array. CAM is content addressable which allows parallel access of multiple memory words.

Associative Memory Organization:
Data stored in an AM are addressed by their contents, so it is called CAM, parallel search memory, and multi-access memory AM has capability of performing parallel search and parallel comparision operation (which is not done by RAM).

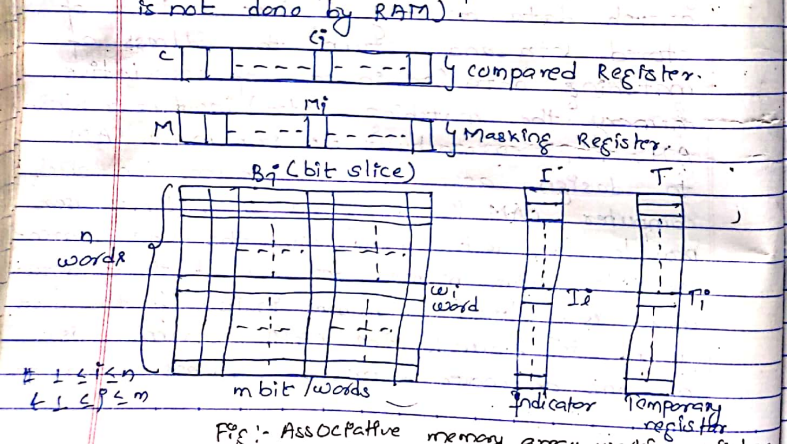


Fig:- Associative memory array working registers

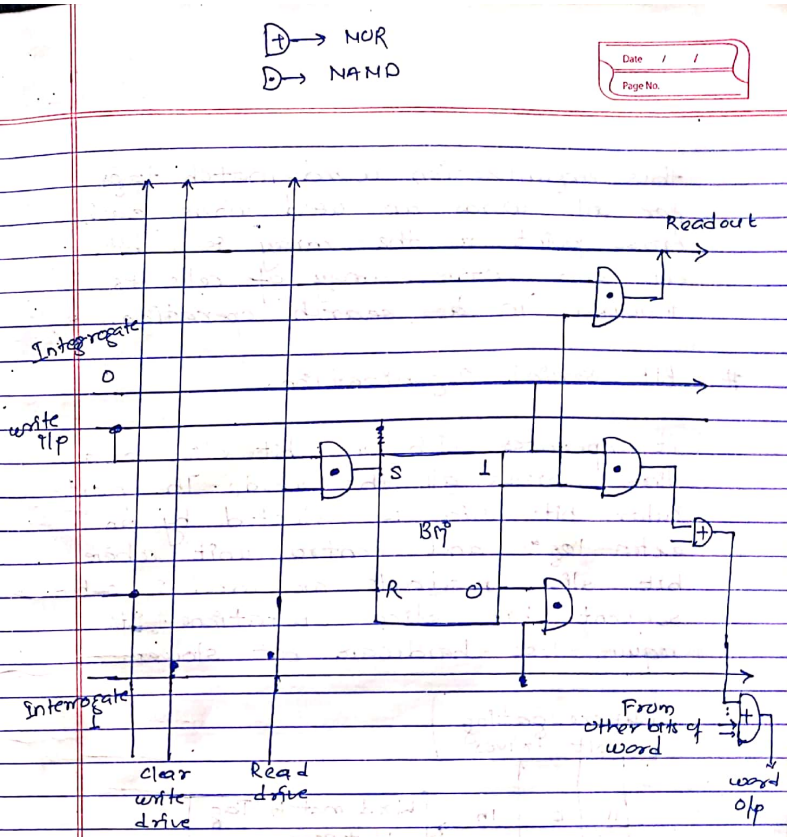


Fig:- Logic design of a typical cell in an AM.

Bit Parallel Organisation.
The comparision process is performed in a parallel by word and parallel by bit fashion and bit slice which are not masked off by the masking pattern are involving in the comparision process. In

this organization word match tags for all words are used where each cross point in the array is a bit cell and entire array of cells is involved in a search operation.

Bit Serial Organization.

It operates with one bit slice at a time across all the words. The particular bit slice is selected by an extra logic and control unit where bit slice readouts are used in subsequent bit slice operations. It require less hardware but slower.

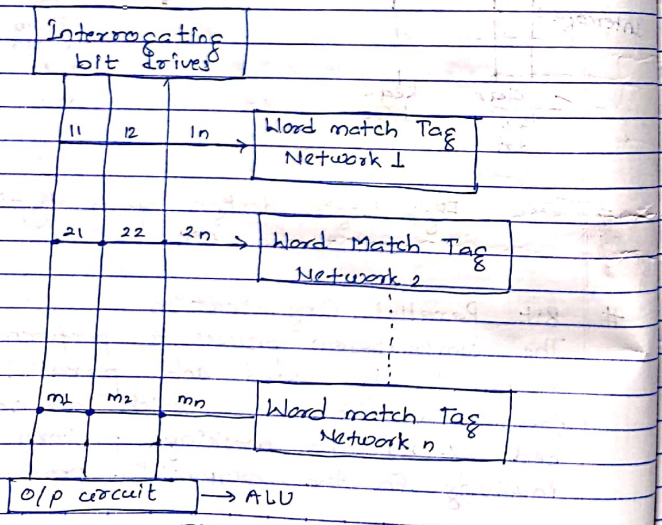


Fig:- Bit parallel organisation
Eg :- PEPE.

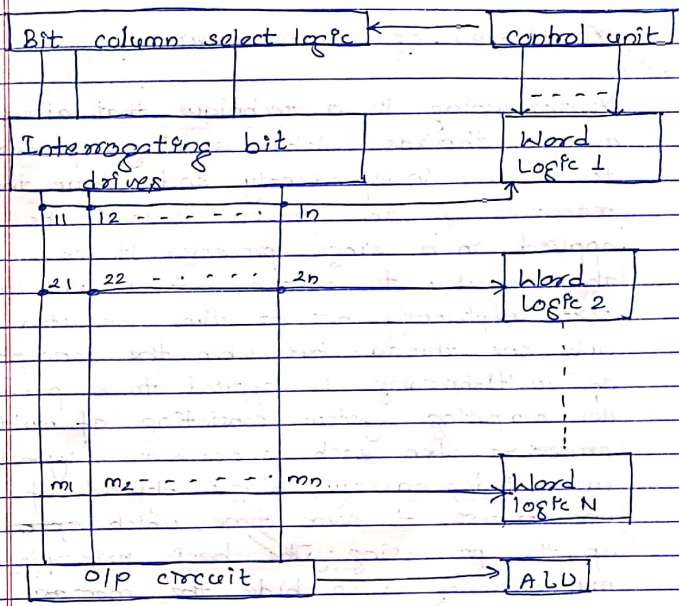


Fig:- Bit Serial Organization
Eg :- STARAN

Multithreading Architecture.

Multithreading is a technique that allows multiple threads within a single process to execute independently and share the resources of the process, when it is applied in a single processor, it causes the processor to switch between the context efficiently but at time executed only one thread. But, when the concept of multithreading is applied to a parallel computing system consisting of multiprocessors then each processor may execute independently, single thread of one process at same time, which speed up the processing. The basic principle of multithreading is to hide the memory latency in scalable parallel system by switching between threads, (when the current thread require a remote memory reference, then instead of waiting for memory access, it switches to execute another thread which avoid memory latency).

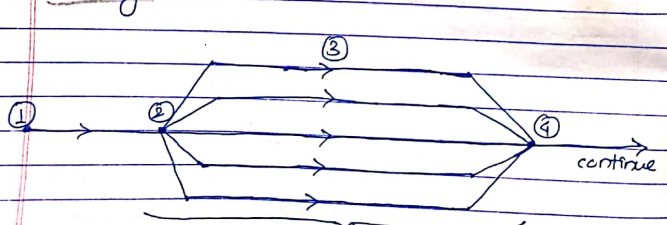


Fig: Multithreading in multiprocessor

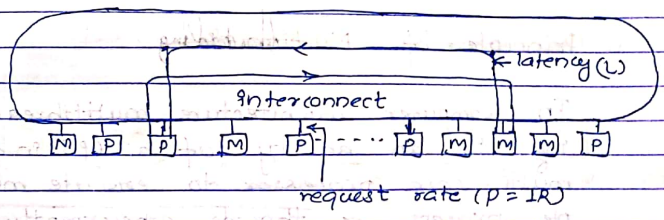


Fig: Architecture Environment

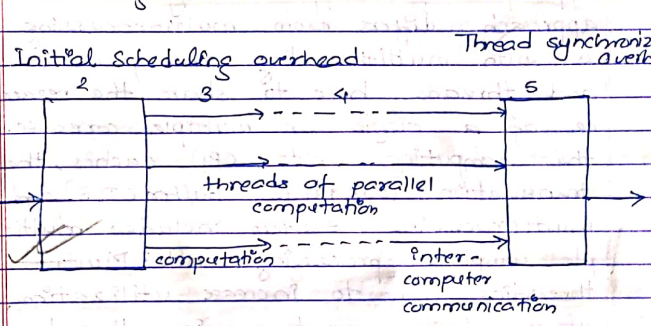


Fig: Multithread computational Model

let us consider one unit of parallel work and a single process may contains many threads.

The processing starts with sequential threads at (1). Then the schedular schedules multiple thread as in (2). and during the concurrent execution of threads they may communicate by passing the message. Finally the multiple threads are synchronized and one unit of parallel work is completed in (4).

Date 08/03/2079
Page No. Thursday

Principle of Multithreading

In computer architecture, multithreading is an ability of a CPU in a multi-core processor to execute multiple process or threads concurrently, appropriately supported by OS. This approach differs from multiprocessing as with multithreading the processes and threads have to share the resource of a single or multiple cores i.e. the computing units, CPU caches, the transaction lookaside Buffer (TLB) and multiprocessing system includes multiple complete processing units. Multithreading aims to increase utilization of a single core by using thread level as well as instruction level parallelism. Since, they are complementary, normally they can be combined in system with multiple multithreading CPUs and CPUs multithreading cores.

Multithreading is accounted because
- Large performance gap between memory and processor.
- Chip consist of many transistor which arise long network latency.
- Multiprocesses on a single chip.

Date / /
Page No.

Parameters to analyse performance of multithreading architecture:

Latency (L)
It is the communication latency on a remote memory access and its value includes network delays, cache miss penalty and delay caused contentions in split transistors. (path)

Number of threads (N)
It is the number of threads that can be interleaved in each processor.

Context-switching Overhead (C) / thread
If refers to cycles lost in performing context switching in a processor and depends on switching machanism.

Interval between switches (R)
It refers to the cycle between switches triggered by remote inference. Inverse $p = \frac{1}{R}$ is called rate of request from remote Access.

In order to increase efficiency
- The rate of request can be reduced by using distributed coherent caches
- eliminate processor waiting through multithreading.

Page No.

Advantages:
1. Faster in overall execution.
2. Prevent Resource to become idle
3. Synchronization in cache values.

Disadvantages:
1. Multiple thread can interfere.
2. Execution of single thread is degraded.
3. Thread scheduling.

Types of multithreading / Instruction:
- Block
- Interleaved
- Simultaneous.

Block Multithreading:
It is the simplest type of multithreading occurs when one thread runs until it is blocked by an event that normally would create a long-latency stall which might be a cache miss that has to access off-chip memory (which may take hundreds of CPU cycle for the data to return). So, instead of waiting for the stall to resolve, a thread processor would switch execution to another thread that was ready to run. where the data for previous thread had arrived, so thread can be placed on the list to run thread.

Date / /
Page No.

Eg:-
Cycle i: Instruction j from thread A is issued.
Cycle i+1: Instruction j+1 from thread A is issued.
cycle i+2: Instruction j+2 from thread A is issued.
cycle i+3: Instruction j+3 from thread A is issued which is a load instruction that misses in all caches.
cycle i+4: thread scheduler is invoked and switches thread B.
cycle i+5: instruction k from thread B is issued.
cycle i+6: instruction k+1 from thread B is issued.

The above example states that Block multithreading is conceptually similar to co-operative multitasking where task voluntarily give-up execution time when they need to wait upon some event. It is also called co-operative or Course Grain Multithreading.

Interleaved Multithreading.
It helps to remove all data dependency stall from the execution pipeline. As one thread is relatively independent from other thread, there is less chance of one instruction

in one pipe stage needing an output from an older instruction in the pipeline. Conceptually, it is similar to pre-emptive multitasking where the time slice is given to each active threads in one CPU cycle.

Eg:-
Cycle i+1 : An instruction from thread A is issued.
Cycle i+2 : An instruction from thread B is issued.

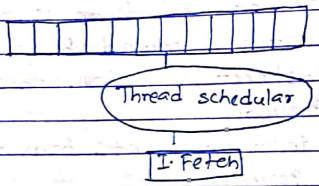


Fig:- Cycle by cycle Interleaved multitasking.

Interleaved, Preemptive, fine-grained or time slice multithreading are more modern technology. In context of Interleaved multithreading it has an additional cost of each pipelines. (Stage tracking thread ID of instruction and also requires large caches and TLBs.)

Simultaneous ~~Multi~~ Multithreading (Threading) / Multitasking.

It is the most advanced type of multithreading which is applied to superscalar processor, where a superscalar processor can issue instruction from multiple threads for every CPU cycle. This helps to exploit parallelism available across multiple threads to decrease the waste associate with unused issue slots.

Eg:-
Cycle i: Instruction j and j+1 from thread A and instruction k from thread B are simultaneously issued.
cycle i+1: Instruction j+2 from thread A, instruction k+1 from thread B & instruction m from thread C are all simultaneously issued.
cycle i+2: Instruction j+3 from A, instruction m+1, m+2 from thread C are all simultaneously issued.

The term 'temporal multithreading' is used to denote when instruction from only one thread can be issued at a time which helps to distinguish other type of multithreading with simultaneous multithreading. Shared resources such as cache and TLBs size must be larger than that of interleaved multithreading as large number of

active threads are processed. Implemented in DEC, EV8, Intel - Hyper threading, IBM, POWER5.

2079/03/13
Tuesday

Latency Hiding
~~Scalable Coherent Multiprocessor with DSM.~~

There are many massively parallel and scalable parallel systems which uses distributed memory architecture, and distributed memory so any Processing Element in multiprocessor system may access the remote memory to fetch the memory data which needs certain interval and ~~this interval and~~ this interval is called memory latency or latency. Delays may be accounted due to interconnection hardware, communication delay and memory response time (these factor causes long latency in accessing remote memory). Memory latency increases because memory is not closed to processor and speed of memory is less than processor, speed of interconnection network between processor and memory is less.

The latency problem can be improved by enhancing their scalability and programmability is concerned with the effective performance of the program which depends on memory latency.

Mechanism to tackle latency,

Latency avoidance Mechanism.
- Reduce cache miss operation.

Latency Reduction Mechanism.
- Reduce by using appropriate communication Hardware and software.

Latency Hiding Mechanism.
It focuses on hiding the latency by the use of overlapping operation during latency i.e. the operation of remote memory access is overlapped with other useful operation by processor so that it need not to wait till the remote data is accessed.

In distributed shared Memory machines, access to remote memory is likely to be slow compared to the ever increasing speeds of processor so any scalable architecture must rely on techniques to reduce or hide remote - memory access latency (RMAL)

S.N

cache

Scalable Coherent Multiprocessor with Distributed Shared Memory (DSM)

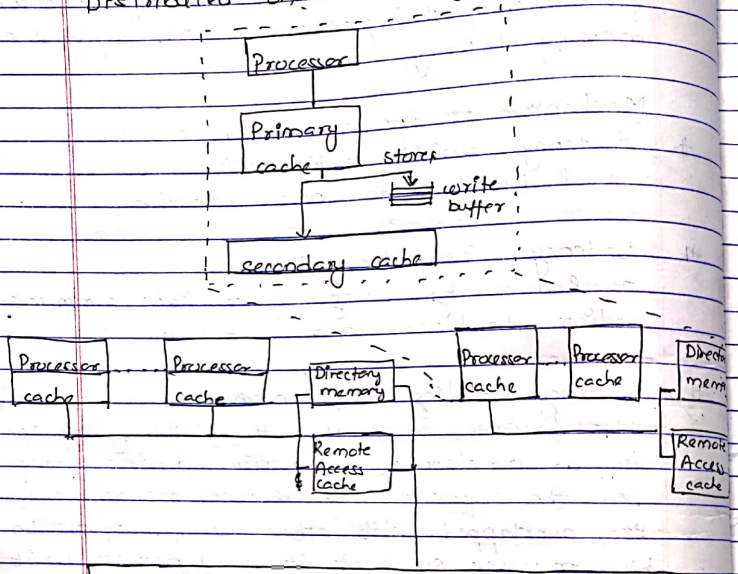
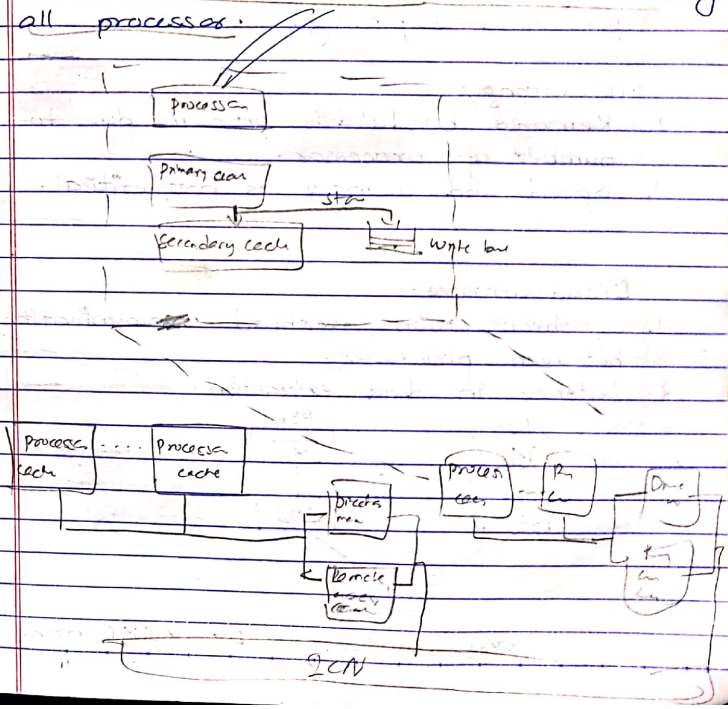


Fig:- Scalable Coherent Multiprocessor.

It is large scale, cache-coherent, NUMA multiprocessor system which consist of multiprocessor clusters connected through a scalable, a low latency interconnection Network. The physical memory is distributed among processing nodes in various clusters and distributed memory forms global memory. Distributed Directory Based Protocol (DDBP) maintains cache coherence where it keeps track of each memory block of remote node and acknowledgement message are used to inform the originating when invalidation is completed. Two level local caches is used and load, write is separated with the use of write buffer. Directory memory and remote access caches are accounted on each cluster and are shared by all processor.



Diff betn Loosely coupled & Tightly coupled.
Imp

Date: Ki 03/2074
Page No. Wednesday

37

Chapter 5:
Distributed Memory Architecture.

The class of DM-MIMD machines is the fastest growing part of high performance computer but it is more difficult to deal with than shared memory machines and DM-SIMD machines. In DM-MIMD, the user has to distribute the data over the processor the processor and also the data exchange the between processor has to be performed explicitly.

Advantages:
1. Removes bandwidth problem due to number of processor.
2. Speed and memory is not critical.

Disadvantages:
1. Synchronization overhead communication between processor.
2. Latency in data extraction.

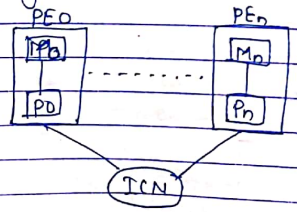


Fig:- Structure of DM-MIMD Archi

Differentiate betn Tightly coupled & loosely coupled architecture / multiprocessor

Date / /
Page No.

Classification:

Basis	Types
System Architecture	client-server and Peer to Peer.
Communication N/w	Bus and Switched
Coupling	Tightly and loosely coupled

Tightly Coupled Architecture.
Non-Uniform Memory Access (NUMA)

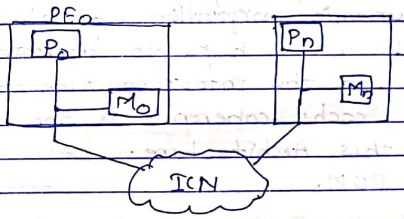


Fig: Structure of NUMA

In this architecture, logically shardded memory physically distributed where there is different access of local and remote memory blocks so there is latency in access of remote memory. It is sensitive to data and program distribution close to distributed memory system but the programming paradigm is different. Eg: Cray T3D.

Cache Only Memory Access (COMA)

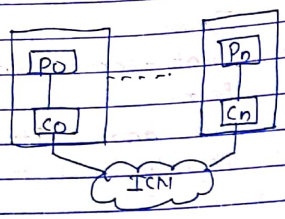


Fig: Structure of COMA.

Each block of shared memory works as local cache of a processors and continuous dynamic migration of data is accounted which helps to decrease the traffic on the interconnection network but cache coherence is the problem of this Architecture.
Eg: KSR-1, DDM.

Cache - Coherent non Uniform Memory Access (CC-GONUMA)
It is the combined form of NUMA and COMA where initially the statistical data is distributed and then dynamically the data is migrated. Cache coherent problem of COMA is solved in this architecture.
Eg: Convex SPP1000, MIT.

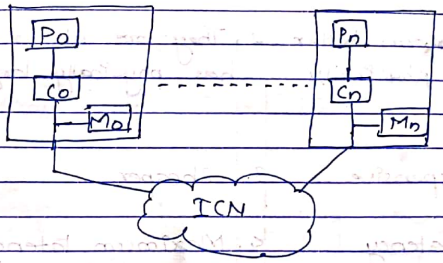


Fig:- Structure of CC-NUMA.

Loosely Coupled Architecture.

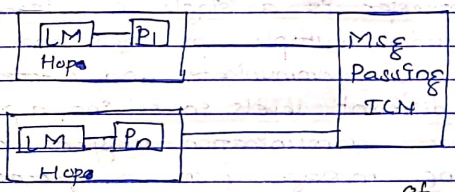


Fig:- Structure of loosely coupled Architecture.

Differentiate Between Tightly coupled and Loosely coupled. architecture / multiprocessor.

Tightly Coupled	Loosely coupled.
1. They contain multiple processor that are connected at bus level where processor may have access to central shared memory	1. They are based on multiple standalone single or dual processor computer interconnected through high speed communication system.

2.	They perform better and are physically smaller.	They are slower and has physically larger.
3.	More expensive	Cheaper
4.	Minimal latency	Maximum latency and low data rate.
5.	Low power consumption	High power consumption

A multiprocessor is a single computer that includes multiple processor and processor may communicate and co-operate at different levels in solving a problem where communication may occur by sending message from one processor by sharing common memory. A multiprocessor system is controlled by OS which provides interaction beth processor and program at process, data set and data element level by coordinating various processor activities either through shared memory or interprocessor message.

Multiprocessor are of two types on the basis of memory organized:

1. Tightly coupled.
2. Loosely coupled.

What are the major three org that can be employed in design of OS for multiprocessor? Explain any two of them.

Tightly coupled:

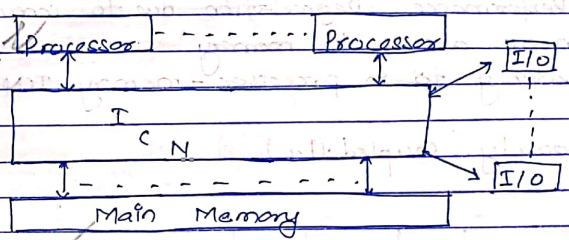


Fig:- Tightly coupled multiprocessor.

In this organization, main memory is shared by all processor which provides high degree of resource sharing and multiple processor communicate through a shared memory. This type of architecture is used to speed up the execution of a single large program in time critical operation. So, the data rate can be increased, where a small local memory or high speed buffer may be used in each processor (CP). There is complete connectivity between processors and memory by inserting an Interconnection Network (ICN). In this architecture main memory, i/o devices, ICN and processors are connected at bus level and entire system is controlled by OS which provides interaction between processor and program at process and data levels.

Explain L.C.A with T.C.A

Disadvantage:

1. Performance Degradation due to concurrent access to memory.
2. Latency to in processer - memory ICN.

Loosely coupled (L.C)

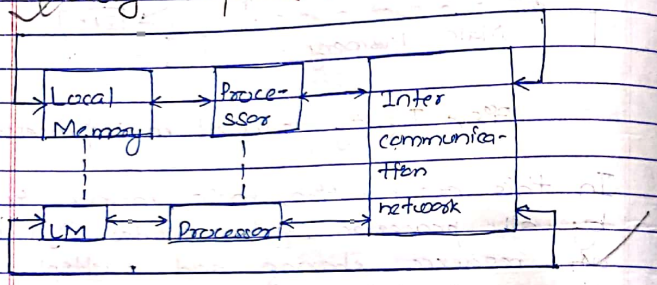


Fig:- Loosely coupled Arch.

In this organization, each processor has its own memory which helps in faster access to its local memory. So, it decrease memory conflicts experienced in tightly coupled. Each processor has its own I/o device and a large local memory to access instruction and data. Two processer interacts with eachother by message passing using standard primitive function send() and receive() and is controlled by Message Transfer System (MTS).

The access to remote memory attached to other processer takes longer time due to delay through ICN, but easier to organise and write os in loosely coupled system.

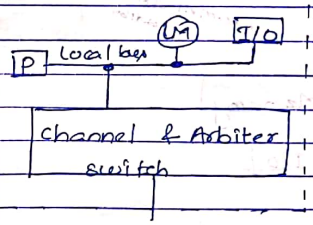


Fig:- Computer Module.

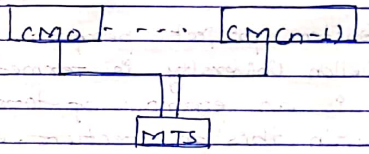


Fig:- Loosely coupled of computer Module.

Date 15.03.2024
Page No. Thursday

Q5) Cluster computing as an application of loosely coupled Architecture.

a) CM* → L.S

b) Hadoop: S.N

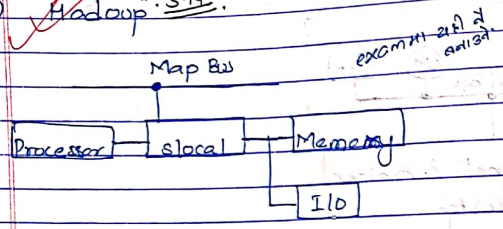


Fig:- Computer Model.

Computer Modules* (CM*)

- A computer system project performed at Carnegie Mellon University is termed as CM* which is an example of lousely coupled system. In this architecture, each computer module consist of local switch called slocal. The slocal interpret and route the processor request to the memory and I/o device outside the computer module via map bus, and also accepts reference from other computer module to its local memory and I/o devices.

Fig: Computer model.

Date / /
Page No.

- The computer module are connected in hierarchical clusters by two-level buses, which can be expressed as,

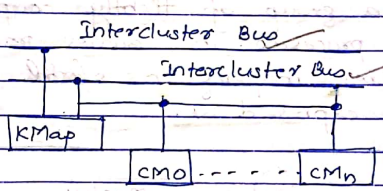


Fig:- A cluster of computer Module.

- A cluster is regarded as the lowest level, and is composed of computer Module (CM), kMap, Map Buses where number of computer module are connected to a map bus so that they share the single kmap. (K map is a process that is responsible for (mapping address and routing data between slocals.

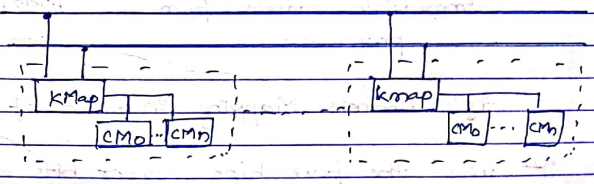


Fig:- Network of CM cluster.

Clustering can enhance the co-operative ability

of the processor in a cluster to operate on shared data with low communication overhead and provides hardware facilities to execute a group of tightly coupled co-operating processes. The address translation or address mapping in slocal of CM* can be expressed as.

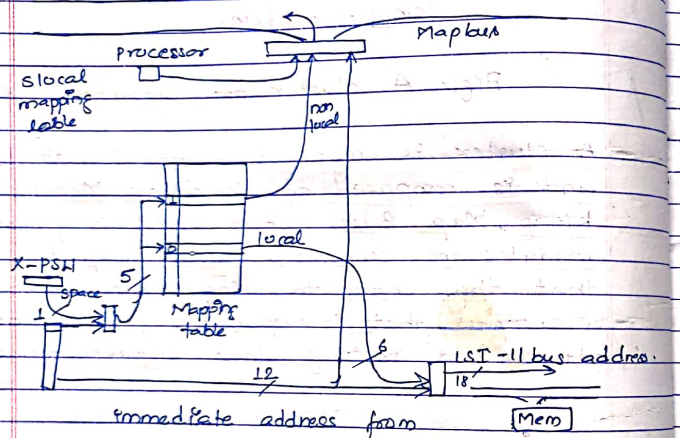


Fig:- Address Mapping in slocal of CM**

It uses the four high order bits of the Processor along with the current address space indicated by x-bit PSW (Processor status word) to access a mapping table which determines whether the reference is local or non-local. Cluster communicate through intercluster buses which are connected between kmaps. The kmap is a microprogrammed with 150ns cycle and three processor complex with common data memory. Three processor are kBus, Linc and Pmap. kBus is bus controller which arbitrates requests to the map bus. Pmap is the mapping processor which responds to request from kbus and Linc. Linc performs most of the request processing.

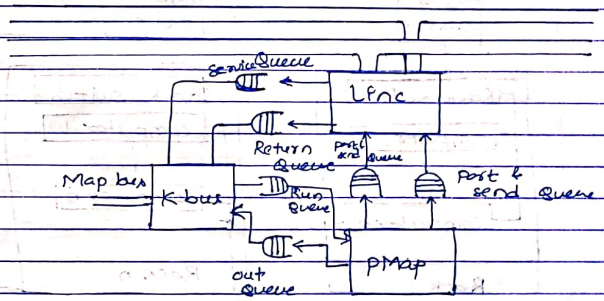


Fig:- Component of kmap

Hadoop.. S.N.

It is a architecture which supports the processing of large data sets in distributed computing environment which consist of Mapreduce, Hadoop distributed File system & related projects. Mapredue & HDFS are map

component of Hadoop. Hadoop cluster is a special type of computational cluster, design for storing and Analysing vast amount of unstructural data in distributed computing environment where cluster run on low cost computers.

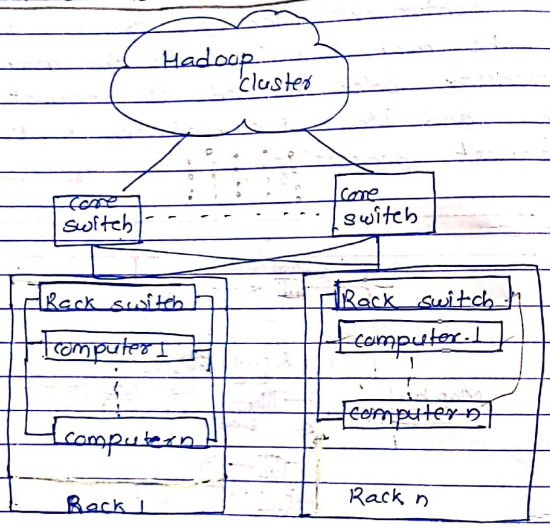


Fig:- Hadoop architecture.

Eg:- Yahoo Hadoop Architecture.

List out various levels of parallelism that can be obtained in context of programmability issue.

Chapter - 6:
Programmability Issue.

Type and level of parallelism.

Bit level parallelism:
From the advent of VLSI computer chip fabrication technology, speed up in computer word size so that more amount of information can be manipulated by processor per clock cycle. Increasing the word size reduces the number of instruction the processor must execute to perform an operation on variable whose size are greater than the length of word bit from each integer & higher bits using an add with carry instruction where it requires two instruction to complete single operations and where 16 bit processor can complete in single instruction.

4 bits → 8 bits → 16 bits → 32 bits → 64 bit

development of processor.

Instruction level Parallelism:
A computer program is a stream of instruction executed by processor. A processor can only issue less than one instruc

tion per clock cycle without instruction, instruction level parallelism for subscalar processor. In instruction level parallelism, the instruction are re-ordered and combined into groups which are then executed in parallel without changing the result of program. Present reordern processors have multi stage instruction pipelines where each stage refers to action to be performed in instruction. A processor with an N-stage pipeline can have upto N different instruction of different stage of completion and thus can issue one instruction per clock cycle.

Fig:- Five stage of RICS Processor
Note:: P4 computer has 35 stage for a instruction.

Task level parallelism:
It is the characteristic of a parallel program that "entirely different calculations can be performed on either the same or different data sets. It explains data parallelism where the same calculations is performed on the same or different sets of data.

It involves the decomposition of a task into subtask and then allocating each sub-task to a processor for execution.

The processor would execute sub-task simultaneously and often cooperating. Task parallelism doesn't usually scale with size of a problem.

come with other question for 6 marks as in SN.

Date 27/03/2021
Page No. Tuesday

Answered

OS configuration for multiprocessor computer OR
OS for parallel processing.

The presence of more than one processing unit in the system introduce a new dimension into the design of operating system. The modularity of processors and the interconnection structure among them effect the system development. Communication schemes, synchronization mechanism and placement, and assignment policies terminate the efficiency of the OS. There are three organization that have been utilized in the design of OS for multiprocessor.
They are:

Master - slave OS.
The executive routine is always executed in the same processor. If the slave needs services that must be provided by the supervisor, then it must request that service and wait until the current program on the master processor is interrupted and the supervisor is dispatched. The supervisor and the routines that it uses, donot have to be reentrant (replicate) since there is only the one processor using them. Having a single processor executing the supervisor simplifies the table conflict and lock-out problem for control tables. The overall system is comparitively inflexible. This type of system requires comparitively simple software and hardware. The entire system is subject to catastrophic failure that requires operator invention to restart when the processor designated as the master has a failure or in recoverable error. Idle time on the slave system can build up and become quite appreciable if the master cannot execute execute the dispatching routines fast encugh to keep the slave busy. This is most effective for special application where the work load is well defined or for asymmetrical systems in which the slaves have less capability than the master processor.

Separate supervisor in each processor.
In this configuration, each processor services its own needs. In effect, each processor (supervisor) has its own set of I/o equipment, files, device etc. Each processor has its own set of private tables, although some table must be common to the entire system and that creates table-access control problems. It is necessary for some of the supervisory code to be reentrant, to provide replicate

separate copies for each processor. The separate supervisor OS is as sensitive as master/slave system, however the restart of an individual processor that has failed will probably be quite difficult so the reconfiguration of I/O usually requires manual intervention and possibly manual switching.

Floating Supervisor OS.
The "master" floats from one processor to another although several of the processors may be executing supervisor service routine at the same time. This type of system can attain better load balancing over all types of resources. Conflict in service request are resolved by priorities that can be set statistically or under dynamic control. Most of the supervisory code must be reentrant since several processor can execute the same service routine at the same time. Table access - conflicts and table lock-out delay can occurs but there is no way to avoid this with multiple supervisor and important point is that they must be controlled in such a way that system integrity is protected.

Date 28/03/2074
Page No. Wednesday

Chapter 7: Program & Network Properties

7.1 Condition of Parallelism

A theoritical treatment of parallelism is needed to build a basis for challenges like computational model, inter processor communication and system integration. Normally parallelism appears in various forms in computing environment which can be attributed to level of parallelism, computational granularity, time and space complexity, communication latencies, scheduling techniques and load balancing. So trade off is accounted between time, space, performance and cost.

7.1.1 Data and Resource Dependencies.
The ability to execute several program statement in parallel requires each segments to be independent of the other segment where independence can be of form as data, control and resource independence. Dependence graph is used to be describe relation where node of the graph is termed as program statement i.e. instruction and directed edge with various labels shows the order relations among instruction. Analysis of graph is done so that parallelization and vectorization could exist.

7.1.2 # Data dependence

Data dependence indicates the ordering relationship between statement. It can be classified as:

a) Flow dependence:
A statement S2 is flow dependent on statement S1 if an execution path exist from S1 to S2 and if at least one output of S1 feeds at input to S2. It is denoted by S1 → S2.

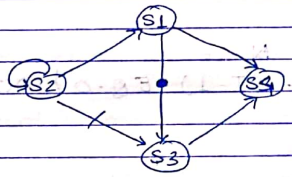
b) Antidependence:
A statement S2 is antidependent on S1 if S2 follows S1 in program order and if the output of S2 overlaps the input to S1. A direct arrow crossed with bar is used to indicate antidependence S1 ↛ S2.

c) Output dependence:
S1 and S2 are output dependences if they produce same output variable. It is denoted as: S1 ⟶ S2.

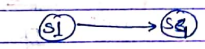
d) I/o dependence:
S1 and S2 are I/o dependent if they reference same file where S1 & S2 are I/o statement.

e) Unknown dependence.
The dependence of two statement cannot be determined when subscript donot contain loop index variable, subscript having different coefficient at loop variable, subscript is non-linear in loop index variable then there occurs known dependence:
Eg:-

S1: Load R1, A
S2: Add R2, R1
S3: Move R1, R3
S4: Store B, R1.



Dependence Graph



Input/output dependence.

The data dependence should not be violated during program execution otherwise extraneous result may be produced with change in program order.

Control dependence:
It refers to the situation where the order of execution of statements cannot be determined before runtime. Eg:- Conditional

LT = less than.
EQ = Equal to.

statement may eliminate data dependencies among instruction. Dependencies may also exist between operation performed in successive iteration of a loping procedure.

Eg:-

```
Do
    I = 1, N
    A(I) = C(I)
    If (A(I) .LT. 0) A(I) = 1
Continue

Do
    I = 1, N
    If (A(I-1) .EQ. 0) A(I) = 0
Continue
```

control Independent

control dependent

Control dependence often prohibits parallelism from being exploited. Compiler techniques or hardware branch prediction techniques are needed to get around the control dependences in order to exploit more parallelism.

Resource Dependence.

It demands the independence of the work to be done which is concerned with the conflicts in using shared resources, conflicting resources in ALU is ALU dependence and in storage is storage dependence.

7.1.2 Bernstein's Condition.

Bernstein revealed a set of condition based on which, two process can execute in parallel. A process is a software entity corresponding to the abstraction of a program fragment defined at various processing levels. The input set I_i of a Process P_i as the set of all input variable needed to execute the process is defined. The output set O_i consist of all outputs variable generated after execution of the process P_i. I/p variables are essentially operands which can be fetched from memory or registers and output variables are the result to be stored in working registers or memory location.

for program write from her.

Let us consider two process P_1 and P_2 with input set I_1 and I_2 and output set O_1 and O_2. Two process can execute in parallel & are denoted as $P_1 \| P_2$ if they are independent and gives valid result.

ie. $I_1 \cap O_2 = \phi$

$I_2 \cap O_1 = \phi$

$O_1 \cap O_2 = \phi$

These three condition are known as Bernstein's condition.

I_i refers I/p set or Read set or Domain
O_i refers o/p set or Write set or Range.

So, interms of data dependence, Bernstein's condition indicates that the two process can be execute in parallel if they are flow, anti and output independent. In general a set of Process $P_1, \ldots, P_n$ can execute in parallel if Bernstein's condition are satisfied on a pairwise basis i.e. $P_1 \| P_2 \| \cdots \| P_n$ iff $P_i \| P_j \;\; \forall i \neq j$

Eg:
P_1: $C = D \times E$
P_2: $M = G + C$
P_3: $A = B + C$
P_4: $C = L + M$
P_5: $F = G + E$

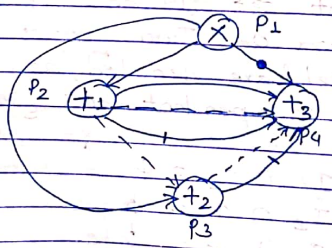
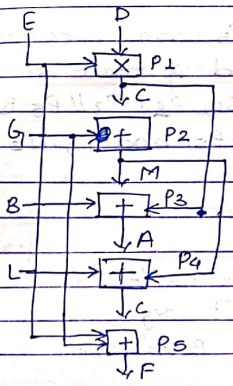
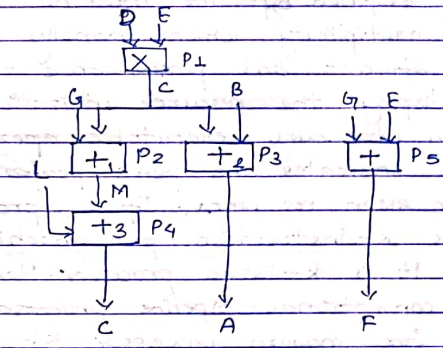


Fig: Data Dependence & Resource Dependence.



Sequential Execution.



Parallel Execution.

If two address are available simultaneously, parallel execution required only three steps. Pairwise there are two pair of statement to check against Bernstein's condition. $P_1 \| P_3$, $P_2 \| P_3$

$P_2 \| P_5$, $P_5 \| P_3$ and $P_4 \| P_5$ can executed in parallel as revealed if there are no resource conflicts. So, $P_2 \| P_3 \| P_5$ is possible as $P_2 \| P_3$, $P_3 \| P_5$ and $P_5 \| P_2$ are possible.

2074/03/29
Thursday.

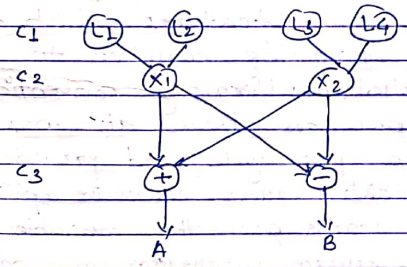
7.1.3 Hardware and software Parallelism.

Hardware Parallelism refers the parallelism defined by the machine architecture multiplicity which normally a function of cost and performance trade off. It displays the resource utilization patterns of simultaneously executable operation which indicates the peak performance of processor resources. If a processor issues k- instruction per machine cycle then it is k-issue processor. Normally, for conventional pipeline processor $k=1$ and for modern processor $k \geq 2$.

Software Parallelism refers the parallelism which is revealed in program profile or in the program flow graph which is normally a function of algorithm, programming style and program design.

The program flow graph displays the pattern of simultaneously executable operation.
Eg:.
Let us consider there are 8 instruction (four loads and four arithmetic operation) to be executed in three consecutive machine cycles. Four load operations are performed in first cycle, followed by two multiply operations in second cycle and add/subtract operation in third cycle. So, average software parallelism is $8/3 = 2.67$ instr/ cycle.

Again, let same program be executed by two issue processor which can execute one memory access and one arithmetic operation simultaneously. So, program must execute in seven machine cycle. Average Hardware parallelism is $8/7 = 1.14$ instr/ cycle.



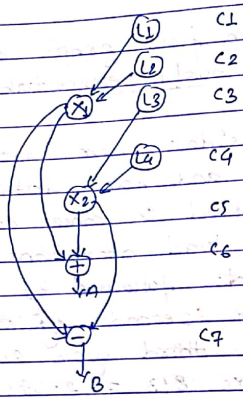


Fig:- Hardware Parallelism.

7.1.4 Role of Compiler.

Compiler techniques are used to exploit hardware feature to improve performance. Loop transformation, software pipelining are some features which are accounted for optimizing compiler to support parallelism. Normally conventional scalar processors issue at most one instruction per cycle and provides few register which causes excessive spilling of temporary results from the available registers. So, software parallelism may not improve performance in conventional scalar processors. There exist a vicious cycle of limited hardware support and use of a naive compiler. So to break cycle, the compiler must be designed jointly with hardware at same time, that leads better interaction between them and provides better solution to mismatch between software and hardware parallelism. And, it also increase flexibility in hardware parallelism and exploits software parallelism in control intensive program. But there exist Hardware and software design trade off in terms of cost, complexity expandability, compability and performance. Multiprocessor has more complexities in compiling than uniprocessor where granularity & communication latency play important roles in the code optimization and scheduling process.

7.2 Program Partitioning and Scheduling.

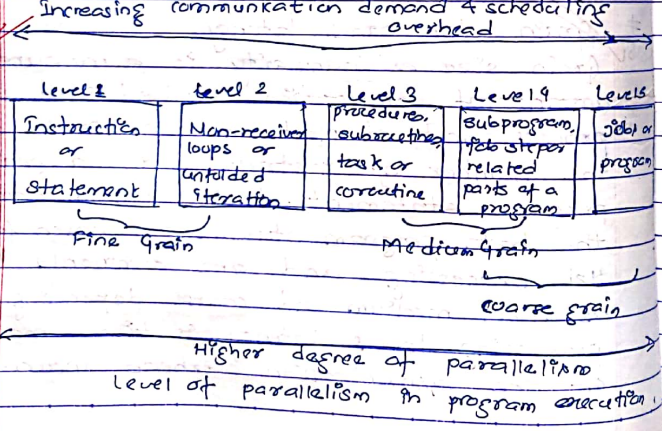
Grain size and latency.

Grain size or Granularity is a measure of amount of computation involved in a software process and simplest measures is to count the number of instruction in a grain i.e. program statement. Grain size determines the basic program statement chosen for parallel processing and are commonly classified as fine, medium and coarse depending on processing levels. Latency is time measures of the communication overhead incurred bet

Grain packing.
Grain size ko level.

machines subsystem.
Eg: Memory latency (time required by a processor to access the memory), synchronization latency (time required for two process to synchronize with each other). Granularity and latency are related with eachother.

Parallelism has been exploited at various processing level of program execution which represents various computational grain sizes and changing communication and control requirement. Generally, execution of a program may involve combination of processing levels which depends on application, formalization, algorithms, language, program, hardware characteristics etc.



Level 1:
Normally it contains less than 20 instruction i.e. fine grain. Fine grain parallelism at this level ranges from two to thousands depending on individual program. A compiler can optimized the exploration of fine grain parallelism and should be able to automatically detect parallelism and translate source codes to a parallel form that can be recognized by run-time system.

Level 2:
It refers to iterative loop ops which contains less than 500 instruction that can be vectorized for pipelined execution or for lock-step execution on SIMD machine of independent in successive iteration. Loop level parallelism is often most optimized program construct to execute on a parallel or vector computer.

Level 3:
It refers to medium grain parallelism of the task, which consist less than 2000 instructions so detection of parallelism is more difficult than fine grain levels.

Level 5:
It refers to the level of jobstep & related sub-program where grain size contains ten to hundred thousand instruction and job steps can overlap across various jobs. Normally in this level parallelism is exploited by algorithm designers or programmers rather than compiler.

level 6:
It corresponds to the parallel execution of essentially independent jobs of a parallel computer where grain size contains more than millions of instructions in a single program. Job level parallelism is handled by the program loader and os where time sharing and space sharing multiprocessor explore the parallelism.

A better performance of computer system can be achieved by balancing granularity and latency.

Grain Packing and Scheduling.
The time complexity involves both computational and communication overheads. The program partitioning involves the algorithm designer, programmer, compiler, OS support.
Eg: Already done

* Fig: Fine grain graph before packing.
# Explaining the figure:

There are 17 nodes in the fine-grain program graph where node 1, 2, 3, 4, 5 & 6 are memory reference operation which takes one cycle to address and six couple to fetch from memory and remaining nodes are CPU operations each requiring two cycle to complete. After packing, the coarse grain node have larger grain size ranging from 4 to 8. Normally, for grain packing fine grain is applied first to achieve a higher degree to parallelism and then combined into coarse grain node to eliminate unnecessary communication delay and reduce overall scheduling overhead. Fine grain partition of a program often demands more interprocessor communication than coarse grained partition so trade off between parallelism and scheduling is accounted.

CPU operation - 2 clock cycle

In exam, 17 [illegible] process [illegible] ....

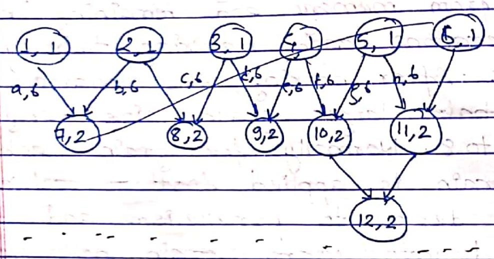
Date 03/04/2024

Var a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q.

Begin

1. a = 1	7. g = a x b	13. m = g x l
2. b = 2	8. h = c x d	14. n = 3 x m
3. c = 3	9. i = d x e	15. o = n x i
4. d = 4	10. j = e x f	16. p = o x h
5. e = 5	11. k = d x f	17. q = p x g.
6. f = 6	12. l = j x k	

Grain size is 2



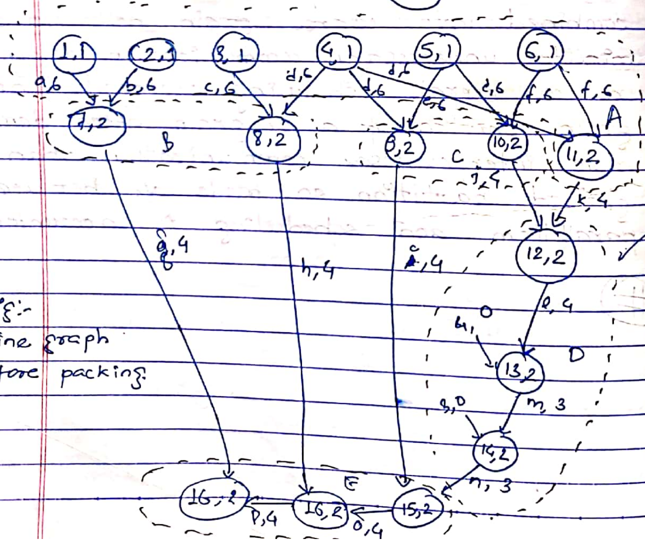


Fig:- Fine graph before packing

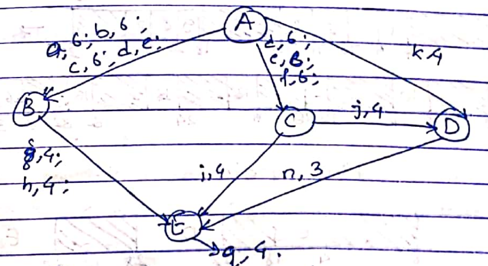


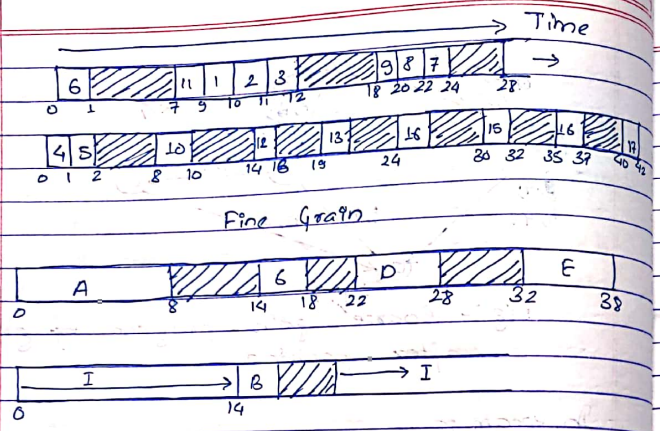
Fig:- Coarse grain program of graph after packing

In exam :-

1. Show list of all
2. Grain size indicate
   Grain size = 1 & 2
3. Fine grain → [illegible]
4. Coarse grain

two times [illegible].

Scanned with CamScanner



Scheduling of fine grain & coarse grain.

7.3 System Interconnect Architecture.
What do you mean by interconnection ~~network~~?

7.3.1 Network Properties and Routing.
The goal of an interconnection network are to provide low-latency, high data transfer rate, wide communication bandwidth where the analysis includes latency bisection bandwidth, data routing function, scalability of parallel architecture. The network are represented by graph with a finite number of node linked by directed or undirected graph.

- Number of Nodes in Graph represents Network Size.
- Number of edges (link or channel) incident on a node represents node degree (d) (in degree and out degree when directed). Node degree reflect number of I/O ports associate with Node and should ideally be small and constant. Symmetric Network are easy to program and implement.
- The maximum number of processor through which a message must pass on its way from source to destination or maximum distance between any two processor in a network is Diameter (D). It helps to measure maximum delay for transmitting message from one processor to another. So, small the diameter better is network topology.
- Arch Connectivity of the Network is minimum number of arch that must be removed from the network to break it into two disconnected network. The arch connecting of some network can be #1 for ring and 2d mesh, #4 for 2d torus, #d for d dimensional hypercube. Larger the arch connectivity lesser than conjuction and better will be network topology.
- Channel width is the number of bits that can communicated simultaneously by a interconnection bus connecting two processor.

- Bisection width is the minimum number of communication links which are removed to divide network into equal halves. It provides the information about the largest number of message which can be sent simultaneously (without needing to use the same wire or routing processor at the same time and so delaying one another) no matter which processor are sending to which other processors. Larger the bisection width is better network topology.

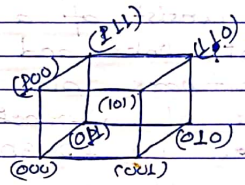
Data Routing function:
A data routing network is used for inter-PE data exchange which is static in hypercube and dynamic in multistage routing network. Shifting, Rotating, Permutation (one to one), Broadcast (one to all), Multicast (Many to Many), Personalized Broadcast (one to many), Shuffle, exchange etc. are routing function.

$ permutation:
Given n objects, there are n! ways in which they can be ordered. It can be specified by giving the rule for reordering a group of objects, which can be used in crossbar switches, multistage network, shifting and broadcast operation. The time complexity increase when value of n increases.

$ shuffle:
The special permutation that entries according to the mapping of the k-bit binary number, $a_{k-1}b\cdots k$ to $bc\cdots ka$ i.e. shifting 1 bit to left and corapping it around to least significant bit position.

Hypercube Routing
If the vertices of a n-dimensional cube are labelled with n-bit numbers so that only one bit differs between each pair of adjacent vertices, then n routing function ~~function routing~~ are defined by the bits in the node address.

Eg:-



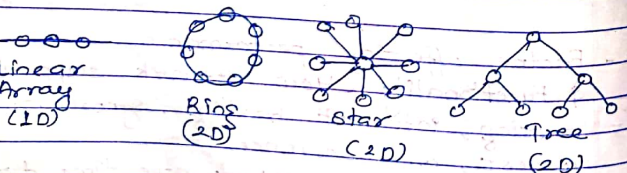
- Factor affecting performance
$ Functionality (How network support data routing, interrupt handling, synchronization request, and coherence)
$ Network latency (Worst case time for a unit

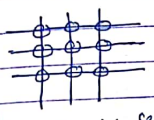
message to be transfered).
$ Bandwidth (maximum data rate)
$ Hardware complexity (implementation costs of wire, logic
$ scalability (how easily does the same adapt to a increasing number of processor, memories etc.)

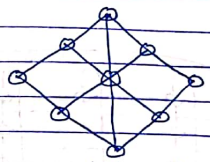
2079/10/04.
Wednesday.

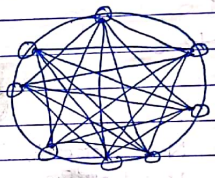
Assignment No. 1

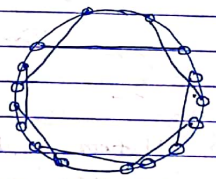
Static Connection Network.
In static network, the interconnection network is fixed and permanent. interconnection path between two processing element and data communication has to follow a fixed route to reach the destination processing element. So, it consist of a number of point to point links and topologies in static network are classified on the basis of dimension required for layout ie. 1D, 2D, 3D or hypercube.

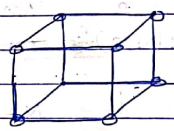


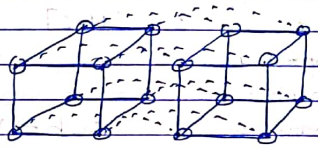

Near Neighbours Mesh (2D)


Systolic Array (2D)

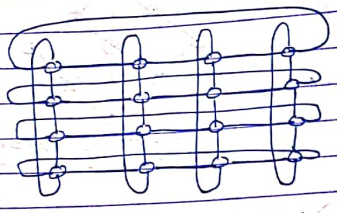

completely connect (3D)


Chordal Ring (3D)


cube (3D)


cube (4D)

Torus architecture is also one of popular network topology which is extension of the mesh topology having wrap round connection. The wraparound connection reduce the torus diameter and at the same time restore the symmetry. It can be of 1D, 2D & 3D. Torus topology is used in Cray T3E.



Torus Architecture.

Network topology	Node degree (d)	Network Diameter (D)	No. of link	Bisection Bandwidth	Symmetry	Network Size
Linear Array	2	N-1	N-1	1	No	N nodes
Ring	2	[N/2]	N	2	Yes	N nodes
Completely Connected	N-1	1	N(N-1)/2	$(N/2)^2$	Yes	N nodes
Binary tree	3	2(h-1) ∵ Tree height $h = \lceil \log_2 N \rceil$	N-1	1	No	N
Star	N-1	2	N-1	[N/2]	No	N nodes
2D-Mesh	4	$2(r-1)$ ∵ $r = \sqrt{N}$	$2N-2r$	r	No	N nodes
2D-Torus	4	$2\lfloor r/2 \rfloor$	2N	2r	Yes	N nodes
Hypercube	n	n ∵ $n = \log_2 N$	nN/2	N/2	Yes	N nodes
Completely connected Chordal	3	$2^{k-1}+1+\lfloor 2^k/2 \rfloor$ ∵ $k \geq 3$	3N/2	$N/(2^k)$	Yes	$N = k \cdot 2^k$

Any four n/w topology and should be explained in table

Q. Distinguish between static & dynamic interconnection n/w.

Date / / Page No.

Ans. L.B. 2015.

Dynamic Connection Network.

The dynamic network are those network where the route through which data move from one PE to another, is established at the time of communication has to be performed, and all processing elements are equivalent distant, and an interconnection path is established when two processing element want to communicate by use of switch. They are difficult to expand than static network. Eg:- Bus based, crossbar, multistage network. Routing is done by comparing bit level representation of source and destination address.

explain more with eg:- 8×8 / 16×16 / 4×4.

classes of dynamic networks.

single stage network
Multistage network

Single stage network.

A single stage switching network with N i/p selector (IS) and N o/p selectors (OS) where at each network stage there is a '1 to D' demultiplexer corresponding to each IS such that $1 < D < N$ and each OS is an 'M to 1' multiplexer such that $1 < M \leq N$. Crossbar network is a single stage network with $D = M = N$. The single stage network is also called recirculating network as in this network connection the

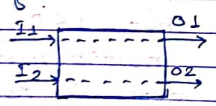
single data items may be have to recirculate several times through the single stage before reaching their final destinations. The number of recirculation depends on the connectivity in single stage network. Normally higher the hardware connectivity lesser is the number of recirculation.

2074/04/05
Thursday,

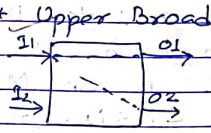
Multistage Network.

Many stage of interconnected switches form a multistage SIMD network. It consist of three feature (switch box, Network structure & control structure. Each box is essentially an interchange device with two i/ps & two output and four possible switch box are:

* Straight

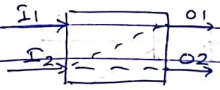


* Upper Broadcast

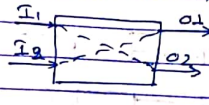


Omega, baseline n/w is only explained in dynamic connection network.

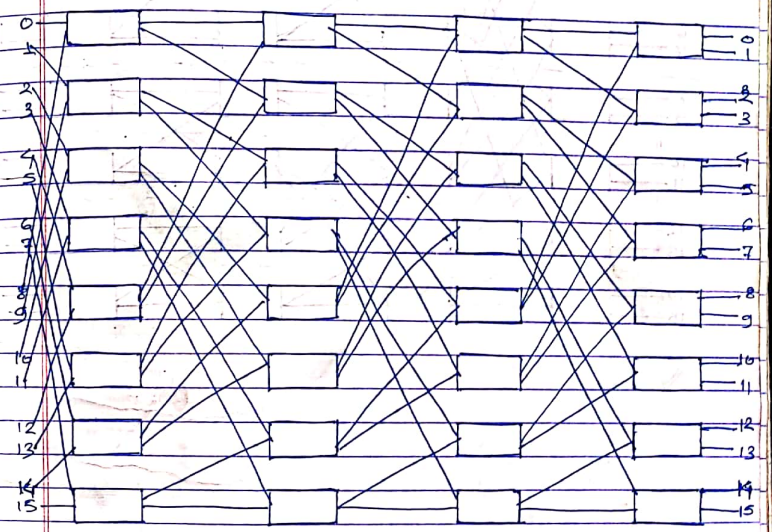
Lower Broadcast



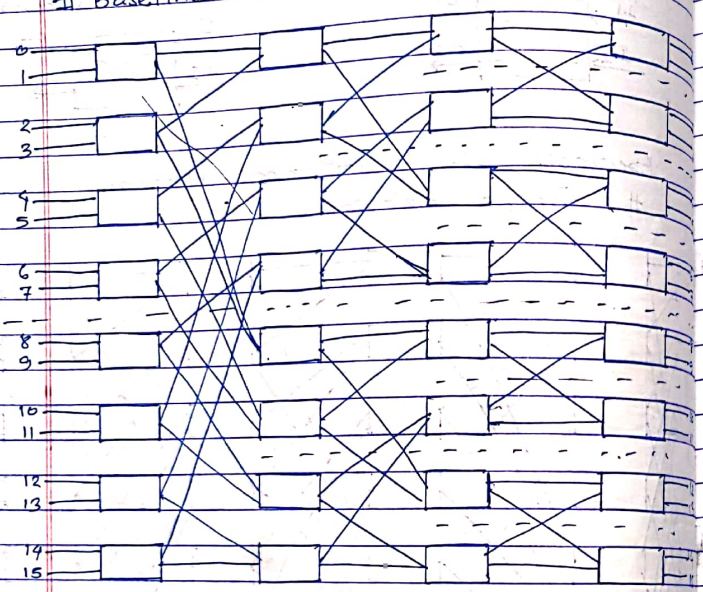
Crossover / exchange



Example of Multistage Network
Omega network.



Baseline Network.



1. a = 1 ∵ Grain size = 1
2. b = 2 ∵ Grain size = 1
3. c = 3
4. d = 4
5. e = 5
6. f = 6
7. g = a x b ∵ Grain size = 2
8. h = c x d
9. i = d x e
10. j = c x f
11. k = d x f
12. l = j x k
13. m = 4 x l
14. n = 8 x m
15. o = n x i
16. p = o x h
17. q = p x g

1 CPU operation = 2 clock cycle.

memory operation

two memory reference + 1 CPU operation = 4.

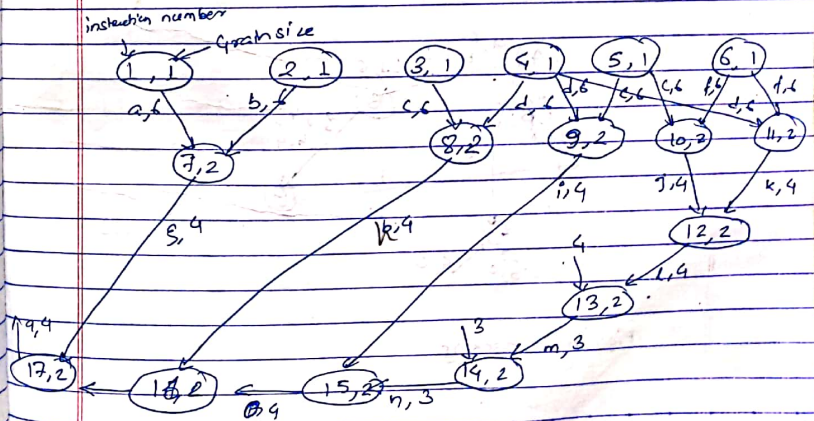
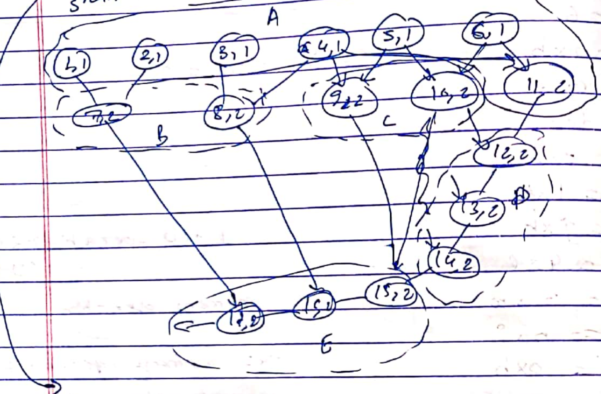


Fig:- Fine grain

the grain ko graph [illegible] ।
explain like, the grain ko graph [illegible]
then, the graph of can be written as:
label [illegible] ।
Grouping of fine grain to form coarse grain.



then, the final graph will be:

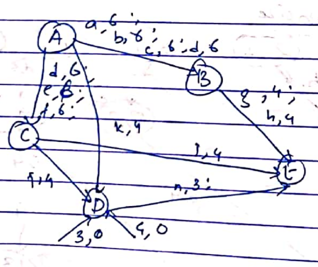


Fig:- Grain packing

After making coarse grain, the execution speed increases as number of process can be run at once like from in coarse grain instructions a, b, c, d, e & f can be performed at once and the values of a, b, c and d can also be sent to coarse grain B.

Differentiate between vector and array Processor with detail explanation.

1. Purpose of Use

Vector Processor

- An approach in which large amount of data ~~presentation~~ processing is represented in one dimensional array of data.

Array processor

- An approach to ~~specify~~ perform a specific operation involving a large processing instead of distributed computing.

2. Way of Processing:

Vector Processor:

Vector Processing can be done as in given example.

Machine level Program.

Initialize I = 0

Begin

Read A[I]

Read B[I]

Store C[I] = A[I] + B[I]

Increment I = 1
If I <= 50 go to Begin
Continue.

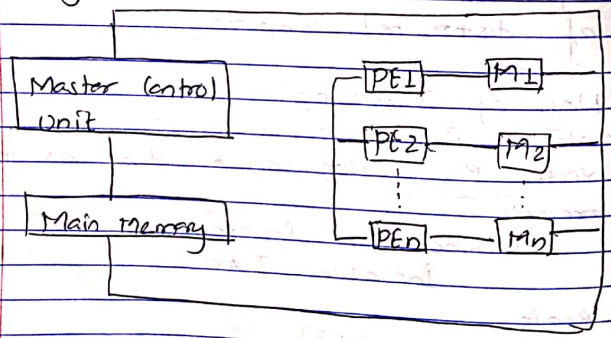
Here, the array are being added using loop. The loop contains 5 statements which repeats to 50 times.
So, the total cycle of CPU is 250 cycle.

Now, using vector processing, unnecessary fetch cycle are reduced as fetch cycles are creation of vectors.

The program can be written as:
C[1:50] = A[1:50] + B[1:50]

Hence, the total cycle of CPU is 50 cycle.

Array Processor.



SIMD Array Processor Technology

Master control unit co-ordinate all the process in the array processor where each processing element have local memory.

Main memory holds the original source data & results obtained from array processor.

Master Process
- Holds a pool of tasks for worker process to do.
- Sends worker a task when requested.
- Collect result from worker

Worker Process
- Get task from master process
- Perform computation
- Send result to master

} repeat.

Advantages:
Vector processing
1) High clock rate
2) Fewer misprediction.
3) Lesser memory access and fast processing time
4) Good memory latency.

Array Processor
1) Faster and scalable.

Demerits:
Vector processor:
1) Not fast with scalar instruction.
2) Expensive and code complexity.

Array
- Highly specialized
- Not widely used.

Application
Vector
1) Super computing
2) Cluster computing
3) Cinema

Array (like systolic array)
1) Matrix Arithmetic
2) Signal processing
3) IP.

Chapter-8 : Parallel Modes, Language and Compilers.

8.2 Parallel languages and compilers.

Programming environment is the group of software tools system softwares support where user need not to program hardwares and focuses only on the program parallelism. So, user need a parallel software environment which can implement parallelism and delay programs.

8.2.1 Language Features for Parallelism

On the basis of functionality, language features of parallelism is classified into 6 types. They are:

1) Optimization features:
It is used for program reconstruction and compilation directives in converting sequential coded program into parallel from which help to match software parallelism with hardware parallelism in machine. e.g:- Automated parallelism.

2) Availability features:
It enhance the user friendliness, make the language portable to a large class of parallel computer and expand applicability of software libraries. It includes scalability, compatibility and portability.

3) Synchronization / communication features:
Single-assigned language shared variables for IPC, logically shared memory, send/receive for message passing, rendezvous in Ada, Remote procedure call (RPC), Barriers etc. are language features for synchronization or for communication purpose.

4. Control of parallelism:
The features involving control constructs for specifying parallelism in various forms are (coarse, medium or fine grains), Grain size, Explicit / Implicit parallelism, Global parallelism, loop parallelism or iteration, task-split parallelism, Divide and conquer paradigm etc.

5. Data Parallelism features:
It specifies how data are accessed and distributed in computers, Run-time automatic decomposition, mapping specification virtual processor support etc. are features included in it.

6. Process management features:
It supports the efficient creation of parallel process, implements multithreadings or multitasking, program partitioning and replication and dynamic load balancing at runtime.

Note:
Language features cannot be implemented without compiler support, O.S. assistance and integration with environment, optimization features emphasizes code parallelism and vectorization at compiler times.

Pic. 2 Parallel language constructs
Let us consider a multidimensional data array indexed by a sequence of triplets one for each dimension.

$e_1 ; e_2 ; e_3$	Here, e_i is a arithmetic
$e_1 ; e_2$	expression which produce a
$e_1 ; * ; e_3$	scalar integer value.
$e_1 ; *$	e_1 is lower bound
e_1	e_2 is upper bound
$*$	e_3 is increment

Eg :- $3(1;4;3;6;8;2;3)$ represents $B(1,6,3)$, $B(4,6,3)$ & $B(4,8,3)$ of three dimensional array.

When the third expression in a triplet is missing a unit stride is assumed. The * notation is the second expression indicates all elements in that dimension starting from e_1 or entire dimension if e_1 is also omitted. When both e_1 & e_2 are omitted the e_1 alone represents a single dimension.

Parallel flow control:
It helps to execute in parallel if there are sufficient processes to handle different iterations however they may be executed serially.

8.2.3 Optimizing compilers for Parallelism

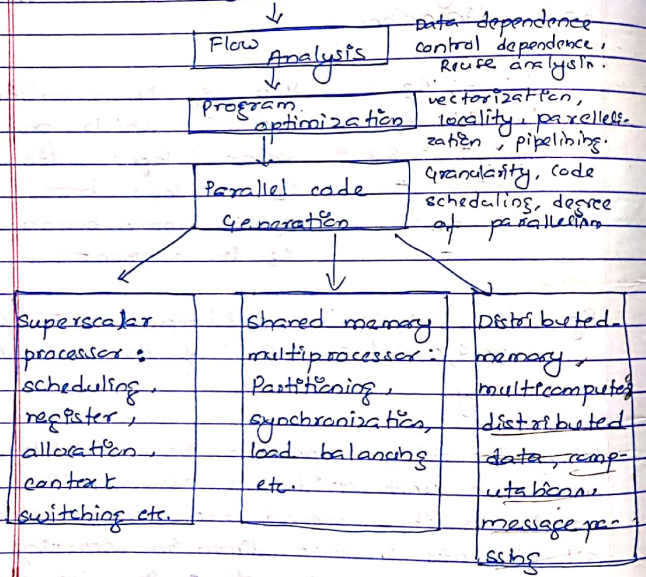


Fig:- Compilation phase for code generation.

The compiler helps to remove burden of program optimization and code generation from the programmer and parallelizing compiler consist of three major phases:

$ Flow Analysis:
It indicates program flowpattern to determine data and control dependence in source code. Flow analysis is conducted at different execution level on different parallel computer. It also indicates data reuse and memory access pattern.

$ Program optimization:
It refers to the transformation of user program to explore the hardware capabilities as much possible where transformation can be conducted at loop level, locality level, or prefetching level with the ultimate goal of reaching global optimization, and transformation must be machine independent. It includes vectorization using pipelined hardware and parallelism using multiple processor simultaneously.

$ Parallel code Generation
It involves transformation from one representation to another called intermediate form. Code generation is closely tied to the instruction scheduling policies used where basic blocks linked by control-flow commands are often

optimized to encourage higher degree of parallelism. Parafrase and Parafrase 2 are two well known compiler developed for optimization.

8.3 Dependence Analysis of Data Arrays.

8.3.1 Iteration Space and Dependence Analysis

The dependence become very hard to determine at compile time as subscript values are not in general available. So, precise and efficient dependence are needed for effective parallelizing compiler. Process of computing all data dependence are in program is dependence analysis.

Dependence testing is the method used to determine whether dependencies exist between two subscripted references to the same array in a loop nest.

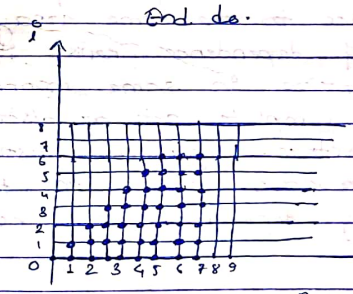
Let us consider to check dependence in statement S_1 & S_2 with n indices as:

```
do i1 = l1, u1
   do i2 = l2, u2
   ....
      do in = ln, un
S1 : A(f1(i1,... in), ... fm(i1,.., in)) = ...
S2 : ... = A(g1(i1, ... in), ... gm(i1,.. in))
      End do
   ....
   End do
End do.
```

Let α & β be vectors of n integers indices within range of the upper & lower bound of the n loops. Then dependence from S_1 & S_2 iff there exist α & β such that α is lexicographically less than or equal to β & satisfies $f_i(\alpha) = g_i(\beta)$, $\forall i, 1 \le i \le m$

The n-dimensional discrete cartesian space for m-deep loops is called an iteration space where iteration are represented as coordinate. Let us consider a two-dimensional iteration space representing two level nested loop with unit increment.

```
do i = 0, 5
   do j = i, 7
      f(i,j)=...
   End do
End do.
```



$0 \le i \le 5$

$i \le j \le 7$

Fig:- A two dimensional space for loopnest.

Sequential order of iteration (is a lexicographic order)

(0,0) (0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)
(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7)
(2,2) (2,3) (2,4) (2,5) (2,6) (2,7)
(3,3) (3,4) (3,5) (3,6) (3,7)
(4,4) (4,5) (4,6) (4,7)
(5,5) (5,6) (5,7)

The above graph shows iteration space of two dimensional discrete cartesian space. If it consist of 3 loops in three dimensional, it also can be represented in three dimensional graph consisting i, j, k as co-ordinate axis.

8.3.2 Subscript Separability and Partitioning

Dependence testing tries to disprove the dependence between pairs of subscripted reference to the same array variable and if dependence exist it tries to characterize in some manner, as a minimum complete set of distance and direction vectors. Dependence testing should be conservative and assume the existence of any dependence it cannot disprove.

~~§~~ subcategories refers to one of the subscripted position in a pair of array reference ie. the pair of subscripts in some dimensions of the two array reference. ~~the pair of subscripts in some dimension~~ The sub-script position are classified by the total number of distinct loop indices they contain - A subscript is said to be zero Index Variable (ZIV) if the subscript position contains no index in either reference, single Index Variable (SIV) if only one index occur in that position and multiple Index Variable (MIV) if it has more than one index

Let us consider an example,

```
do i1 = L1, U1
   do j = L2, U2
      do k = L3, U3
         A(5, i+1, j) = A(N, i, k) + C
      End do
   End do
End do
```

When testing for a flow dependence between two reference to A in the code, the first subscript is ZIV as 5 & N are both constant, the second is SIV as only one index i appears in dimension and third is MIV as both indices j & k appears in third dimension.

S.N

$ Subscript separability

Subscript position is separable in multi-dimensional array if its indices do not occcur in the other subscript. If two different subscript contains the same index, they are called coupled. Separability is vital as multidimensional array references can cause imprecision and dependence testing and if subscripts are separable it helps to compute the direction vector for each subscript independently and merge the direction vectors on a positional with full precision.

let us consider an examples

```
do i1 = l1, u1
   do j = l2, u2
      do k = l3, u3
         A(i,j,k) = A(i, j, k) + c
      Enddo
   Enddo
Enddo
```

The first subscript is separable as index i doesnot appear in other dimensions, but second and third are coupled i.e. A[j,j] because they both contain j. ZIV subscript are separable as they has no index.

$ subscript partioning:

All subscript in a pair of array reference are to be classified as seperable or as part of some minimal coupled group. Separable subscript and coupled group are disjoint set of index. Once partition is achieved, and each partition may be tested isolation and the resulting distance or direction vectors merged without any loss of precision. let us consider loop nest where first subscript yields the direction vector (<) for the i loop.

```
Do i = l1, u1
   Do j = l2, u2
      A(i+1, 5) = A(i, j) + c
   Enddo
Enddo
```

Here j donot appear in any subscript so it is assumes full set of direction vector for the j-loop : i.e. (<), (=), (>) and merging yields { (<,<), (<,=), (<,>) }.

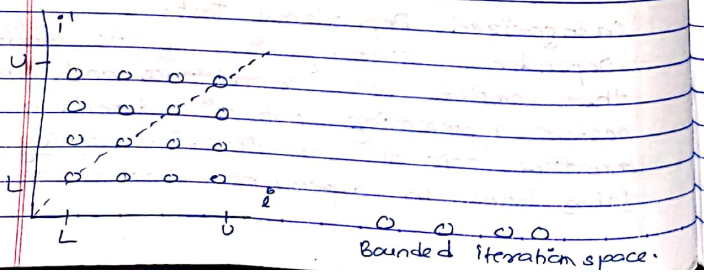
8.3.3.

Categorized Dependence Test.

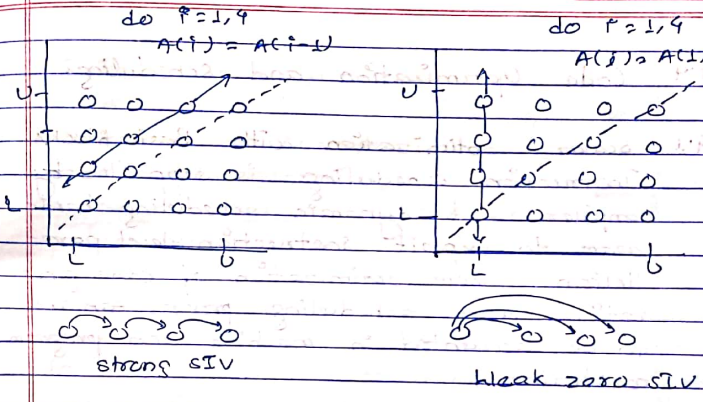
Dependence Testing is used to construct conduct the complete set of distance and direction vector representing potential dependence bet'n arbitary pair of subscripts. The testing algorithm helps in dependence testing based on a

partitioning approach which can isolate unrelated indices and localize the computation involved, Algorithm procedure can be expressed as,

- Partition the subscript into seperable & minimal coupled groups using subscript partitioning algorithm.
- Lable each subscript as ZIV, SIV or MIV.
- Apply appropriate single subscript test based on complexity of the subscript, on separable subscript.
- Apply multiple subscript test on coupled group to produce set of direction vectors.
- If any test yields independence; no dependence exist
- Else, merge all direction vector computed into a single set of direction vector for two references.

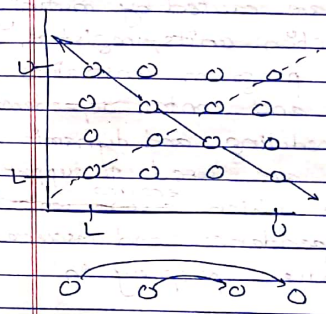


Bounded iteration space.



Strong SIV

Weak zero SIV



Weak - crossing SIV

Fig: Geometric view of SIV test of categorized dependence test.

8.4 Code Optimization and Scheduling.

8.4.1 Scalar optimization with Basic Blocks

Instruction scheduling is supported by compiler and dynamic scheduling hardware to exploit instruction-level parallelism where optimization of code generation and scheduling process can are to be accounted in machine and program constraint.

Machine constraints are caused by mutually exclusive use of function units, registers, datapaths etc. and program constraints are caused by data and control dependence. static instruction scheduling and dynamic instruction scheduling are used to support instruction scheduling which ensures access control dependence, data dependence and resource limitation to be handled properly in concurrent execution.

$ Precedence constraints:
If a flow dependence is detected the write must proceed ahead of read operation involved, output dependence produce different result if two write to the same location are executed in a different order, scalar data dependence is much easier to detect.

$ Basic Block Scheduling.
A basic block is a sequence of statements satisfying the two properties
a) No branches into the middle of block.
b) All statement are executed consecutively if first one is.

An extended blocks are defined for local optimization, as a sequence of statement in which the first statement is the only entry point. While a program is being compiled, basic block records should kept pointers to predecessor and successor storage reclamation techniques or linked list structures can be used to represent blocks.

8.4.2 Local optimization and Global optimizations.

2015

Local optimization are code optimization performed only with in basic blocks. The information needed for optimization are gathered from a single basic block not from extended basic block, No control flow information between block is considered. Some of local optimization are:

1) $ local common subexpression elimination.
If a subexpression is to be evaluated more than once within a single block, it can be replaced by a single evaluation.

2) $ Local constant Folding or propagation.
The initialization of blocks generally computes the constant at compile time, so compiled time generate constants are folded to eliminate unnecessary calculation at run time. In other case a local copy may be propagated to eliminate unnecessary calculation.

3) $ Algebric optimization to simply expression.
Eg:- Replace identity statement
A = B + 0, A = B * 1 to A = B

Commutative law
A = C + D, B = D + C

Associative and Distributive law
(a-b)+c by a-(b-c) if (b-c) has been evaluate.

4) $ Instruction reordering
Code reordering helps to maximize the pipeline utilization or to enable overlapped memory access, to better scheduling and pipeline prevention or memory delay.
I_1 : Load R_1, b
I_2 : Load R_2, B
I_3 : Add R_2, R_1, R_2
I_4 : Load R_3, C

Here I_3 can be delayed which experience no delay.

5) $ Elimination of Dead code or unary operator.
Codes segment / Basic blocks which are not accessible or will never be reaffered can be eliminated to save compile time, run time and space requirements. Unary operators can be eliminated by applying algebraic laws, such as $x + (-y) = x - y$, $-(x-y) = x - y$ etc.

Global optimization are code optimization performed across basic block boundaries where control flow information among basic blocks is needed. Interprocedural global optimization are:

(1) $ Global version of local optimization
It includes global common subexpression elimination, global constant propagation, dead code elimination etc.

(2) $ Loop optimization
It includes various loop transformation to be described in subsequent section for the purpose of vectorization, parallelization & code me & induction variable elimination can simplify loop structure.

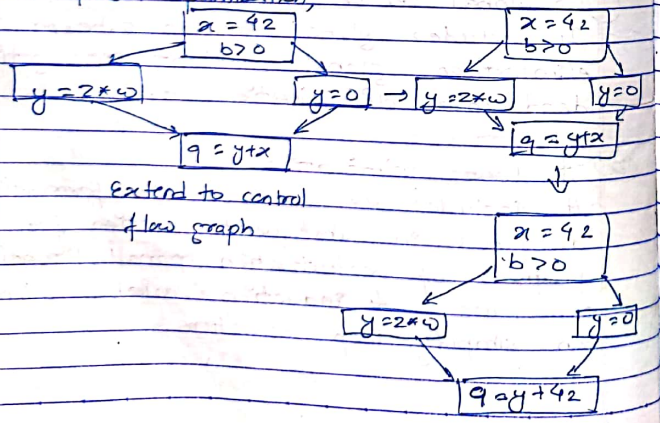
3) $ control flow optimization:
It deal with control structure not directly with loops. Code hoisting eliminates copies of identical codes on parallel paths in a flow graph which save space but may have no impact on execution time.

Eg:
Let x = 42
y = 2*w
q = y+x

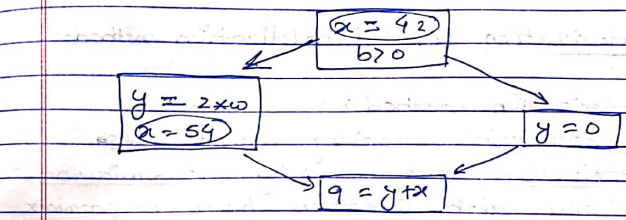
In local optimization,

x = 42 → x = 42 → y = 2w → y = 2*w
y = 2*w → (copy constant propagation) → q = y+42 → (Elimination of dead code) → q = y+42
q = y+x

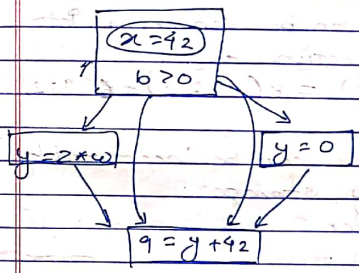
In Global optimization,



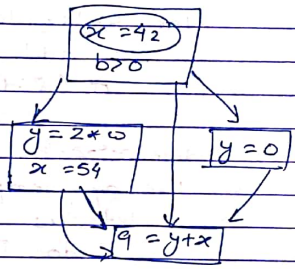
extend to control flow graph



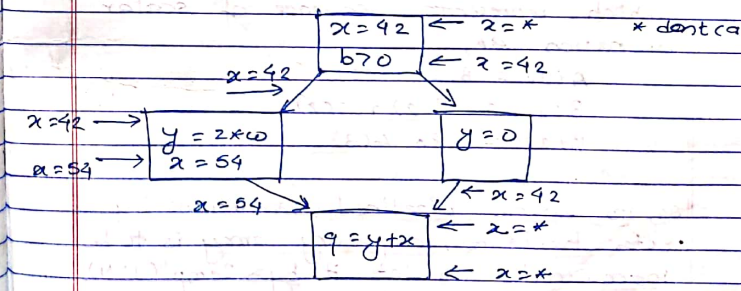
Globally propagate constant is correct?



Revisited.



Revisited



Final condition

What is vectorization & parallelization? What are the various methods of vectorization? Explain any two methods with an eg.

8.4.3. Vectorization and Parallelization methods

Vectorization method:
An optimizing compiler which does vectorization automatically or semiautomatically with direction from programmer is vectorizing compiler or vectorizer.
let us consider an example

```
for (I = 8; I ≤ 120; I+2)
    A[I] = B[I+3] + C[I+1]
```

The scalar loop can be converted into vector add instruction as

A(8:120:2) = B(11:2) + B(9:2)

Vectorization method can be accounted by:

a) Use of temporary storage
let us consider

```
do I = 1, N
    A(I) = B(I) + C(I)
    B(I) = 2 * A(I+1)
```

which represents sequence of scalar operation as,

```
A(1) = B(1) + C(1)
B(1) = B(2) + C(2)
B(2) = 2 * A(3)
```

So, to enable pipelined execution by vector hardware, temporary array is to be introduced and let it be Temp(1:N).
So, it becomes

```
Temp(1:N) = A(2:N+1)
A(1:N) = B(1:N) + C(1:N)
B(1:N) = 2 * Temp(1:N)
```

b) Loop interchanging
let us consider

```
DO I = 2, N
    DO J = 2, N
        A(I,J) = (A(I,J-1) + A(I,J+1))/2
    continue
continue
```

Interchanging loops we get,

```
DO J = 2, N
    DO I = 2, N
        A(I,J) = (A(I,J-1) + A(I,J+1))/2
    continue
continue
```

Now, inner loop I can be vectorized with zero distance vector, as

```
DO J = 2, N
    A(2:N,J) = (A(2:N,J-1) +
               A(2:N,J+1))/2
continue
```

c) Loop distribution:
Nested loops can be vectorized by distributing the outermost loop and vectorizing each of the resulting loops.
Eg:-

```
DO I = 1, N
    B(I,1) = 0
    DO J = 1, M
```

```
A(I) = A(I) + B(I,J) * C(I,J)
       continue
D(I) = E(I) + A(I)
       continue
```

The loop I is distributed to three ~~op~~ copies, seperated by nested loop I form the assignment to array B and D & vectorized as

```
B(1:N, 1) = 0
Do(I = 1, N
   A(I) = A(I) + B(I, 1:M) *
          C(I, 1:M)
Continue
D(1:N) = E(1:N) + A(1:N)
```

d) vector reduction
e) Node splitting
f) other vector optimization

$ parallelization
It spreads a single program onto many threads for parallel execution by multi-processer which reduce the total execution time. Each thread is a sequence of instruction that must execute on a single processor.

8.4.4 Code ~~Optimization~~ Generation and Scheduling. Computation performed are represented by Directed Acyclic Graphs (DAG). The nodes of DAG represents values, edge represent data dependencies. let us consider an example of a block as,

```
t8  = j - 1
t9  = 4 * t8
temp = A[t8]          } temp = A[j-1]

t10 = j + 1
t11 = t10 - 1
t12 = 4 * t11
t13 = A[t12]          } A[j+1]

t14 = j - 1
t15 = 4 * t14         } A[j] = A[j+1]
A[t15] = t13

t15 = j + 1
t16 = t16 - 1
t18 = 4 * t17         } A[j+1] = temp.
A[t18] = temp
```

P.T.O

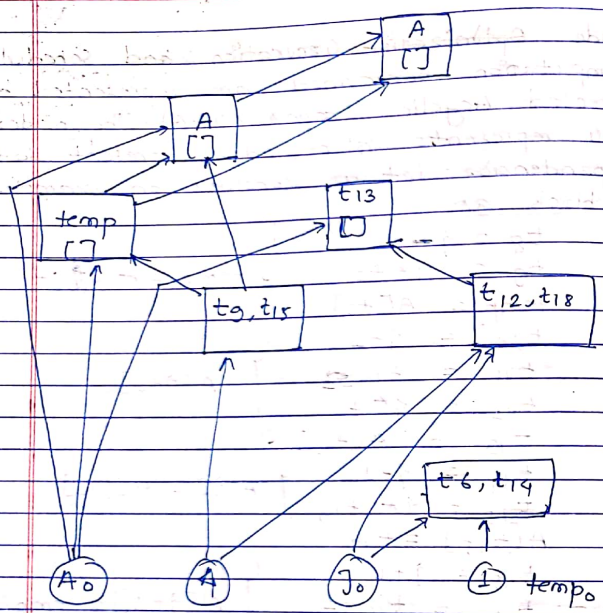


Fig:- DAG.

code scheduling can be conducted by
$ list scheduling
$ cycle scheduling.

8.1 Parallel programming models:

A programming model is a collection of program abstraction, providing a programer a simplified and transparent view of the computer software/hardware system.

They are specifically designed for:
1) Multiprocessor
2) Multicomputers or
3) Vector SIMD computers.

Five models are characterized below for these computers that exploits parallelism with different execution paradigm:

1) Shared-Variable Model:
In the shared-variable model, parallelism depends on how IPC (interprocess communication) is implemented.

In fig a below, shared-variable IPC demands the use of shared memory and mutual exclusion among multiple process accessing the same set of variables.

IPC is implemented in two ways:
1) IPC using shared variable.
2) IPC using message passing.

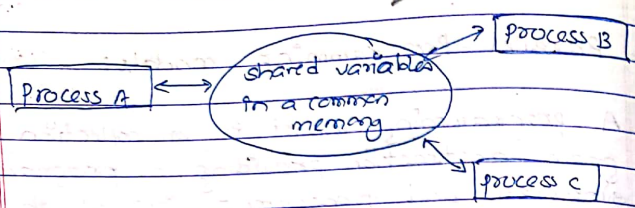


Fig a(i) IPC using shared variable

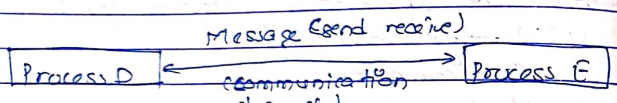


Fig a(ii) IPC using message passing

Fig (a) : Two basic mechanism for IPC.

following are some issues of shared-variable model:

1) Shared-variable communication:
2) Critical section (CS):
   - Code segment accessing shared variables
   - Requirements are:
     1) Mutual exclusion
     2) No deadlock in waiting
     3) Non-preemption
     4) Eventual entry
3) Protected Access:
   - Based on CS value

- Four operation model used:
  1) Multiprogramming
  2) Multiprocessing - two types
     - MIMD mode
     - SIMD mode
  3) Multitasking
  4) Multithreading

4) Partitioning and Replication:
- Program partitioning is a technique for decomposing a large program and data set into many small pieces for parallel execution by multiple processors.
- Program replication refers to duplication of the same program code for parallel execution on multiple processor over different data sets.

5) Synchronization and Scheduling:
Scheduling of divided program modules on parallel processor is much complicated than scheduling of sequencial program or uniprocessor.

Two types:
1) Static scheduling: Conducted at post-compile time.
2) Dynamic scheduling: catches the run-time conditions.

6) Cache coherence and protection:

If the value is returned on a read instruction is always the value written by the latest write instruction on the same memory location is called coherent.
- Multiprocessor cache coherence, sequential consistency model, strong & weak consistency model.

2) Message passing Model:

IPC using message passing model:

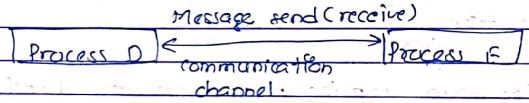


Fig: IPC using message passing.

Message may be:
- Instructions data
- Synchronization or
- Interrupt signals

Communication delay caused by message passing is more than that caused by shared variable.

Techniques for message passing:

1) Synchronous message passing
- It synchronizes the sender and receiver process with time and space. Just like telephone call.
- No shared memory
- No need of mutual exclusion
- No buffer are used in communication channel.
- It can be locked by channel being busy.

2) Asynchronous message passing:
- Doesnot need to synchronize the sender and receiver in time and space.
- Non-blocking can be achieved here.
- Buffers are used to hold the message along the path of connecting channel.
- Message passing programming is gradually changing once the virtual memory from all nodes are combined.

Issues:
- How to distribute or duplicate the program codes and data sets over the processing nodes.
- Tradeoff between computation time and communication overhead must be considered.

3) Distributing the computations:
- Subprogram level is handled rather than

at the instructional or fine grain & process level in a tightly coupled multiprocessor.

3) Data-Parallel Model:
1 - SIMD Lockstep operations
2 - Parallelism explicitly handled by hardware synchronization and flow control.
3 - Synchronization done at compile time rather than run time.
4 - Data parallel programs require the use of pre-distributed data sets.
5 - Choice of parallel data structures is an important considerations.
6 - Emphasize on local computation and data routing operations.
7 - Can be applicable to fine-grain problems.
8 - Implemented either on SIMD or SPMD.
9 - Leads to high degree of parallelism involving thousands of data operations concurrently.

→ Following are some issued handled:
1) Data Parallelism
- Illiac-IV processing model
- Connection Machine CM-2 processing Model
- Synchronous SIMD programming v/s Asynchronous MIMD programming

- SIMD characteristics for data parallelism.
a) Scalar operations and scalar data operands.
b) Vector operations and vector data operands.
c) Constant data operands
d) Masking operations
e) Data-routing operations

2) Array Language Extensions:
- Languages
- Fortan 90 array notation
- CFD for Illiac-IV
- DAP Fortan for AMT Distributed Array Processor (DAP)
- C* for TMC Connection Machine etc.

- Expected characteristics.
- Global Address space
- Explicit data routing among PEs
- No. of PEs be the function of problem size rather than machine size.

3) Compiler support
- Array language expressions and their optimizing compilers must be embedded in familiar standards, such as Fortan and C with an idea to:
1) Unify program execution model
2) Facilitate precise control of massively parallel hardware.

3) Enable incremental migration to data parallel execution:

- Array sectioning
  - Allows a programmer to reference a section of a region of a multidimensional array designated by specifying a:
    - start index
    - Bound (or upper limit)
    - stride (or step-size)

- Vector valued subscripts are often used to construct arrays from arbitary permutations of another ~~array~~ array.

4) Object oriented model:
- Object are created and manipulated dynamically.
- Processing is performed using object.
- Concurrent programming model are built-up from low-level object such as processes, queue and semaphore into high-level objects like monitors and program module.

① Concurrent OOP achieve parallelism using three methods:
- Pipeline concurrency.

- Divide and conquer concurrency.
- Co-operating problem solving.

② An actor model:
- Presented as framework for COOP
- They are self-contained, interactive, independent components of a computing system.
- Basic primitives are: create to, send to, became.

③ Parallelism in COOP:
- 3 patterns:
  1. Pipeline concurrency
  2) Divide and conquer concurrency.
  3) Co-operative problem solving.

① Concurrent OOP - 3 application demand:
1) There is increased use of interacting processes by individual users.

2) Workstation n/w have become a cost-effective mechanism.

3) Multiprocessor technology in several variants has advanced to the point of providing supercomputing power.

Date / /
Page No.

5) Functional and Logic Model:

- Two language oriented programming for parallel processing are purposed.

1) Functional programming model:

- It emphasizes functionality of a program.
- No concepts of storage, assignment and branching
- All single-assignment and dataflow languages are functional in nature.
- Eg:- Lisp, SISAL and Strand 88.

2) Logic programming model:

- Based on logic programming that are suitable for dealing with large database.
- Adopts an implicit search strategy and support parallelism in logic inference process.
- Eg: ① Concurrent Prolog - developed by Shapiro (1986) and
② Concurrent Parlog - introduced by Clark (1987) are two programming language.