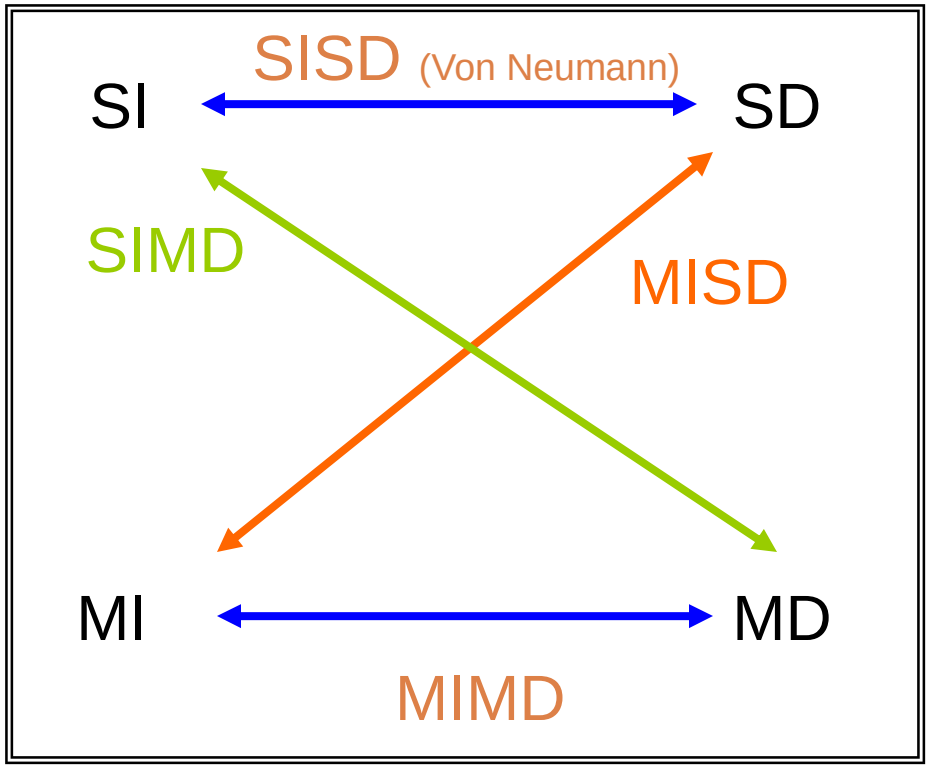
PARALLEL MACHINES AND COMPUTATIONS.

The Evolution of Parallel Computers

- Sequential Model => one instruction at the time.
- Need for multiple operations on disjoint data items simultaneously.
- Flynn classification of the concurrency of data
  - Single Instruction (SI)
  - Multiple Instruction (MI)
  - Single Data (SD)
  - Multiple Data (MD)

The Evolution of Parallel Computers

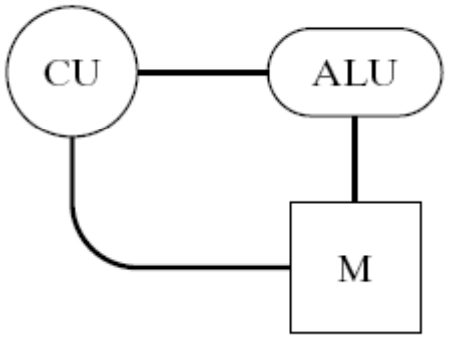
Flynn Classification combination



The Evolution of Parallel Computers
Von Neumann Architecture

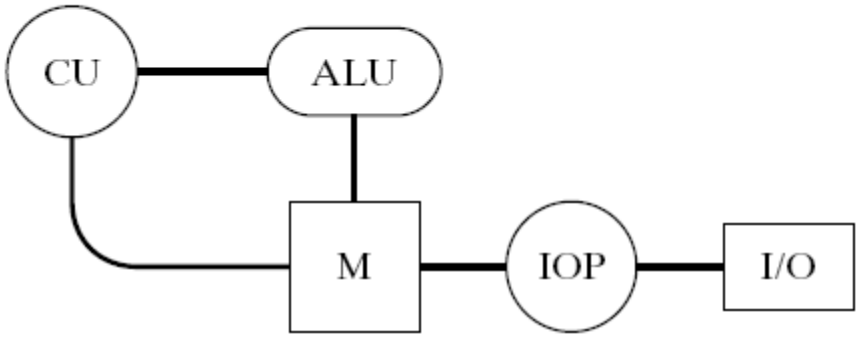
- Instruction Fetch
- Instruction Decode
- Effective Operand calculation
- Operand Fetch
- Execute
- Store Result

SISD (Von Neumann)



The Evolution of Parallel Computers
Von Neumann Architecture

Improved SISD (Von Neumann)



- IOP provide concurrency between fast I/O and CPU
- Multiplexing CPU between programs to maximize CPU idle time.
- Interleaving
- Pipelining (Overlapping instructions)

The Evolution of Parallel Computers

Overlapped fetch/execute cycle for an SISD Computer

Improved SISD (Von Neumann)

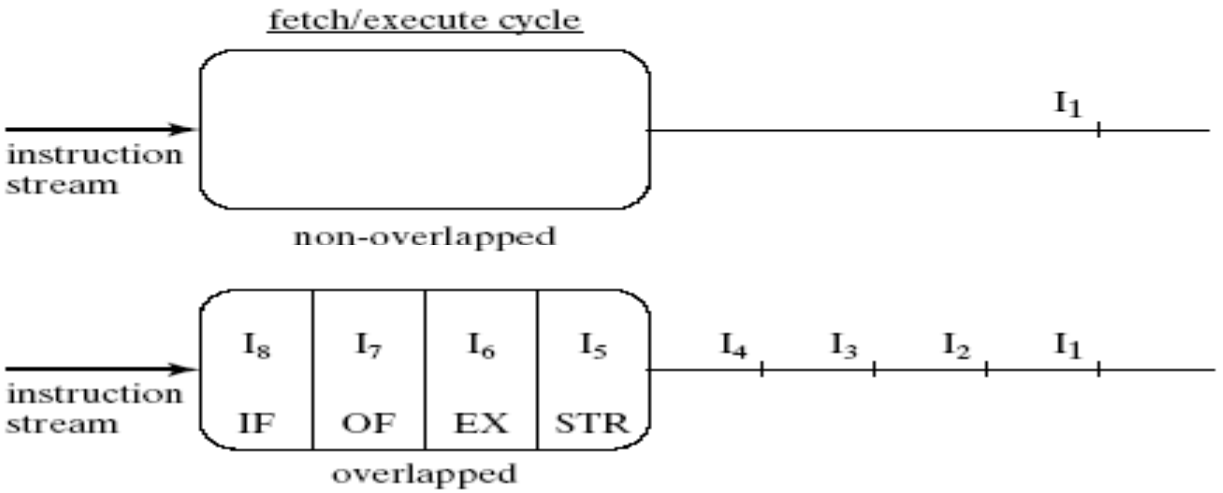
Instruction fetch	I1	I2	I3	Jump	–	–	–
Operand fetch	–	I1	I2	I3	Jump	–	–
Execute	–	–	I1	I2	I3	Jump	–
Result store	–	–	–	I1	I2	I3	Jump
Time	1	2	3	4	5	6	7

- Pipeline: Technique to overlap operations and introduce parallelism.
- Pipeline *start-up* time for the example takes four time units.
- One instruction is executed every time unit.
- Pipeline complexities or hazards may delay the pipeline.
- Pipeline Flush

The Evolution of Parallel Computers

Overlapped fetch/execute cycle for an SISD Computer

Improved SISD (Von Neumann)



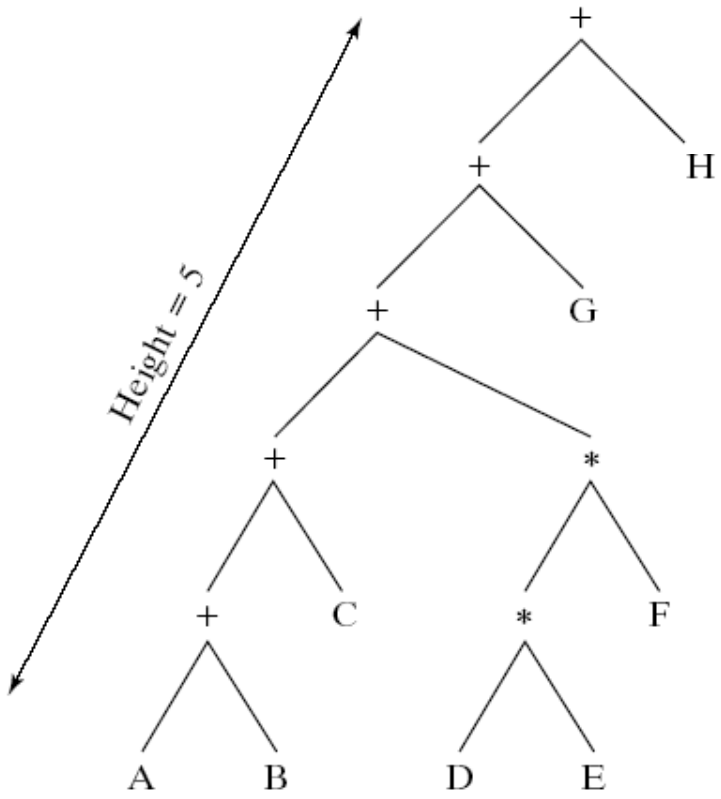
Pipeline improvement Techniques and problems:

- Two instruction Buffers.
- Multiple Arithmetic Units.
- Lookahead
- Scoreboarding
- Resource Conflict
- Output Dependance

The Evolution of Parallel Computers

Height Tree Evaluation (In Order)

$$A + B + C + (D \times E \times F) + G + H.$$



$S_1: T_1 = A + B$

$S_2: T_1 = T_1 + C$

$S_3: T_2 = D * E$

$S_4: T_2 = T_2 * F$

$S_5: T_1 = T_1 + T_2$

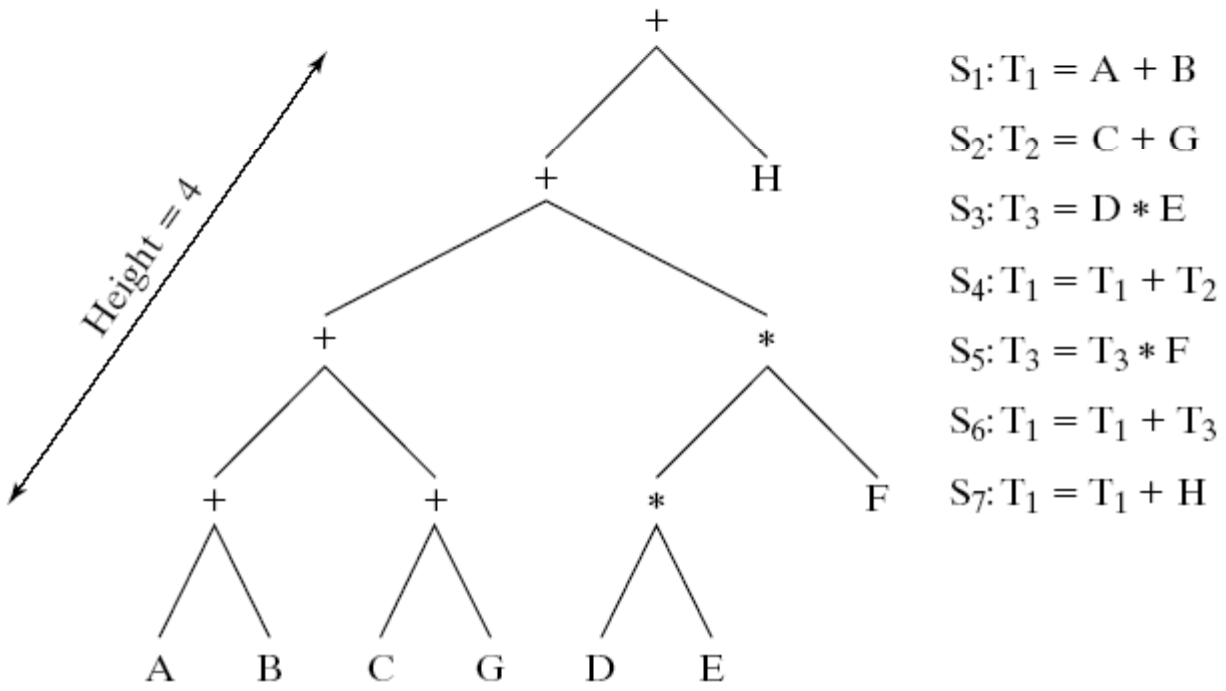
$S_6: T_1 = T_1 + G$

$S_7: T_1 = T_1 + H$

The Evolution of Parallel Computers

Height Tree Evaluation (Reorganized)

$$A + B + C + (D \times E \times F) + G + H.$$



S_1: $T_1 = A + B$

S_2: $T_2 = C + G$

S_3: $T_3 = D * E$

S_4: $T_1 = T_1 + T_2$

S_5: $T_3 = T_3 * F$

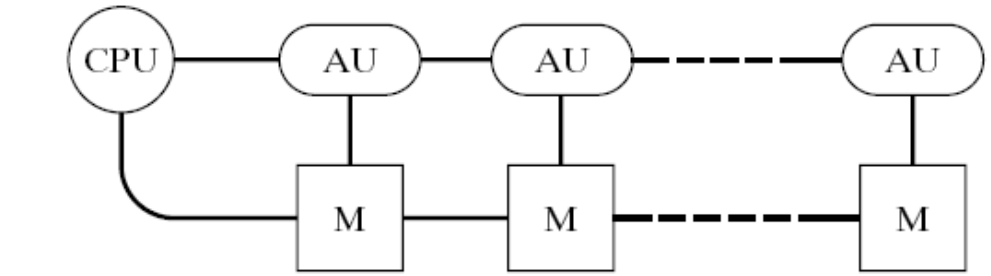
S_6: $T_1 = T_1 + T_3$

S_7: $T_1 = T_1 + H$

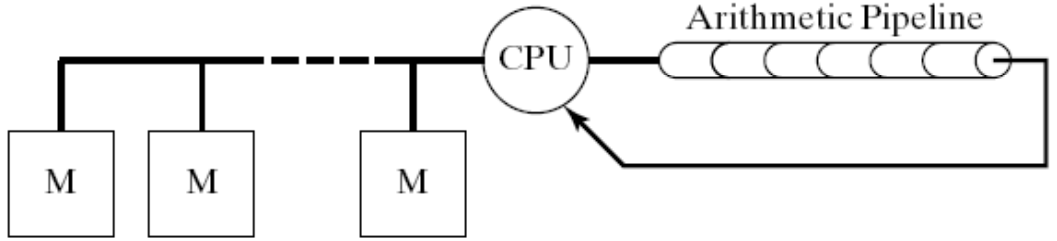
- Reordering instruction execution is necessary for better parallelism

The Evolution of Parallel Computers

Vector SIMD Computers



(a) True SIMD or vector computer



(b) Pipelined SIMD computer

AU — Arithmetic unit CPU — Central processing unit M — Memory

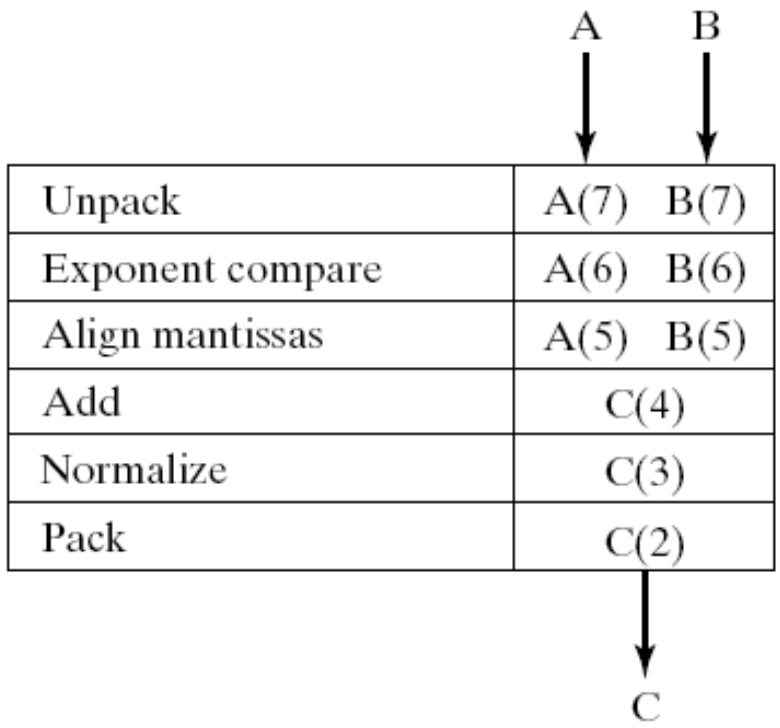
Figure 1-8

True and pipelined SIMD architectures.

- Single Instruction on Multiple Data (SIMD)
- Vector Operations: Repetitive operations applied to different data groups

The Evolution of Parallel Computers

SIMD Floating Point Addition Pipeline



- A Pipeline can keep the amount of parallel activity high while significantly reducing the hardware requirement.
- The front end of the pipeline is not constrained to its length.
- **Startup Cost:** Empties after each vector operation (Flush) and fills when a vector operation starts.

The Evolution of Parallel Computers

SIMD Computers

- Multiple pipelines used to enhance speed.

- Scalar Arithmetic units overlapped.

- The move form True to Pipelined SIMD machines is dictated by the cost/performance ratio and the flexibility of added vector length.

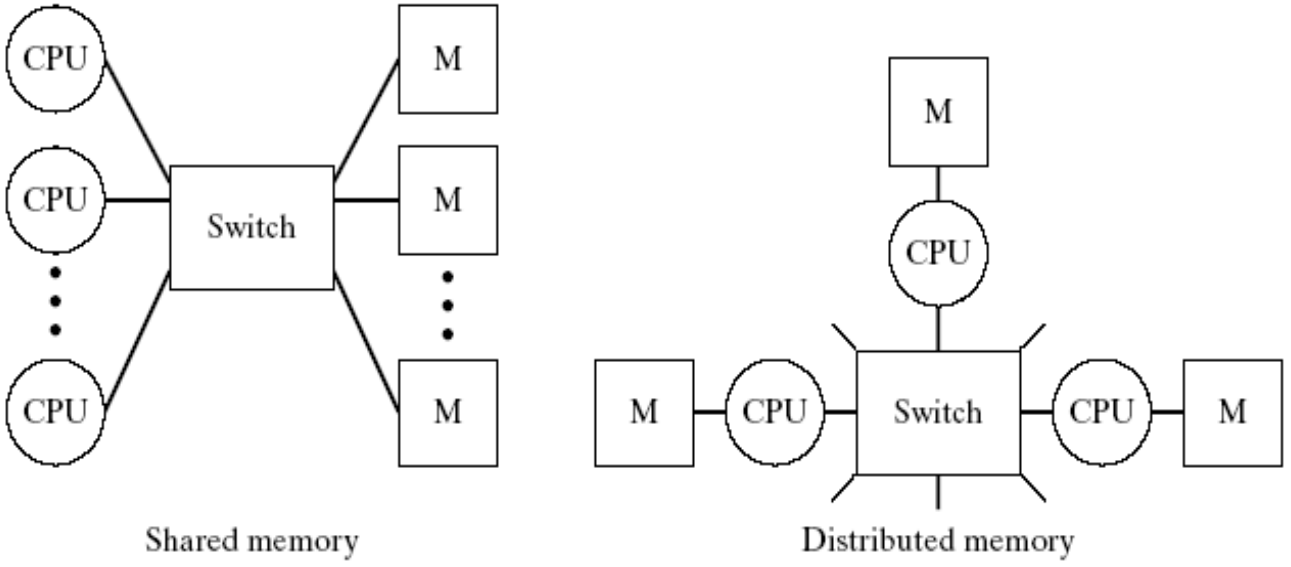
- Numerous arithmetic units of a TRUE SIMD machine are partialy used fro short vectors.

- SIMD computers in detail in Chapter 3.

The Evolution of Parallel Computers

MIMD Computers

- Multiple instruction streams active simultaneously.
- Two prototypical forms of multiprocessor:
  - –Shared Memory
  - –Distributed Memory

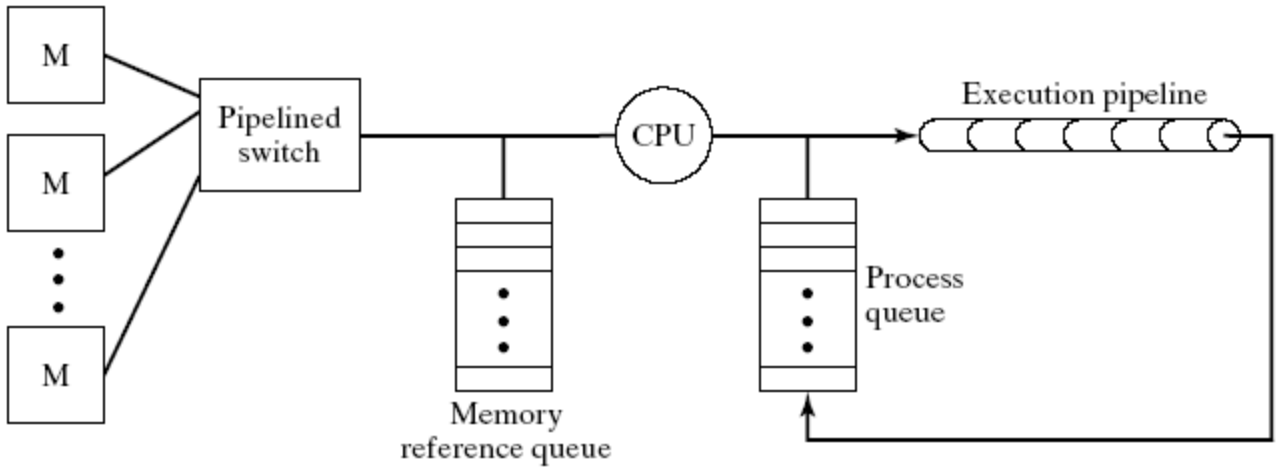


(a) True MIMD or multiprocessor

The Evolution of Parallel Computers

MIMD Computers

• True and Pipelined architectures
•Multiple sequential programs in parallel increase throughput.
•Multiple processors execute different parts of a single program to complete a single task faster.
•Cooperation between programs or shared resources resides in shared memory.
•Inter-process communication regulates message passing.



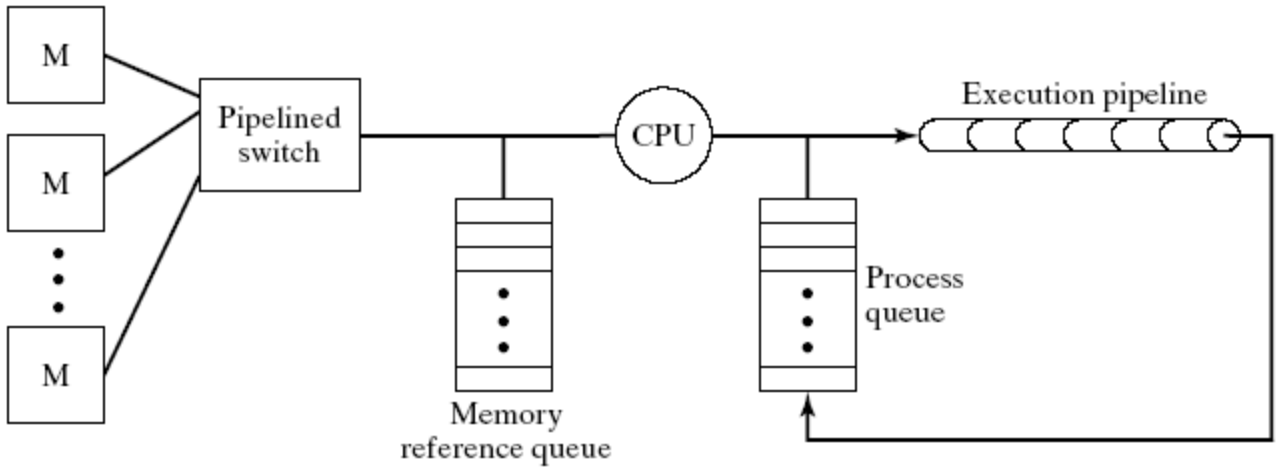
(b) Pipelined MIMD computer

CPU — Central processing unit M — Memory

The Evolution of Parallel Computers

MIMD Computers

•Multiple instruction streams supported by pipelining rather than separate complete processors.
•Reduced hardware and increased flexibility
•Pipelined MIMD machines = Multithreaded Computers



(b) Pipelined MIMD computer

CPU — Central processing unit M — Memory

The Evolution of Parallel Computers

MIMD Computers

•Instructions in a pipeline come form different processes and often need no pause in execution after each process is completed.
•Synchronization failure: Explicit synchronization is needed between instruction streams that may cause a pipeline delay.

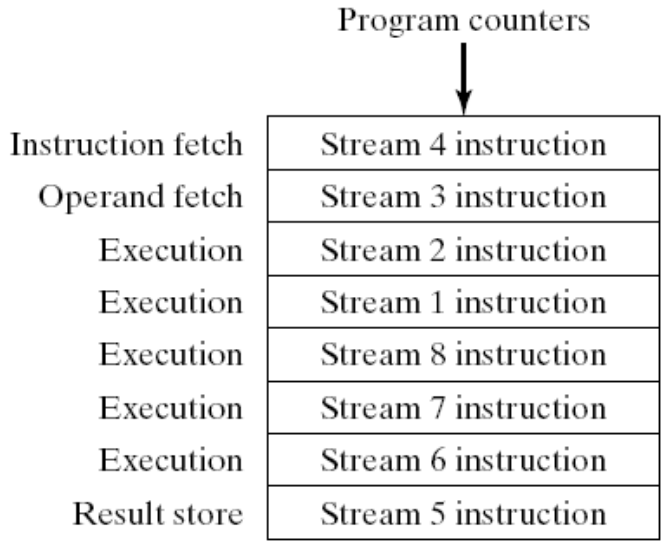


Figure 1-11
Pipelined fetch/execute cycle in shared-memory MIMD.

The Evolution of Parallel Computers

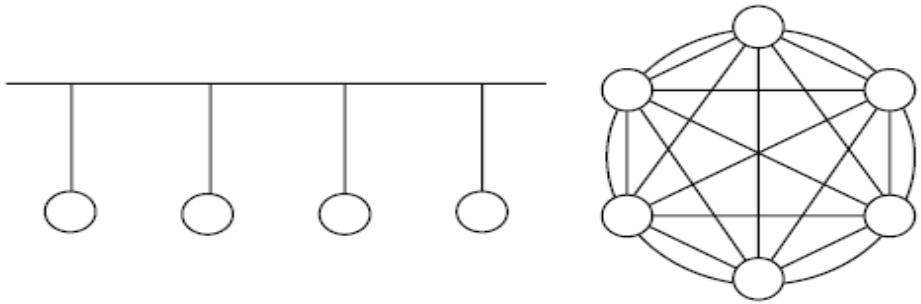
Interconnection Networks (IN)

•Interconnection Network is the connectivity provided to facilitate routing data between the parallel components of a specific architecture.

• TRUE SIMD, arithmetic units use IN to route data to the right processing elements.

•Pipelined SIMD, IN permits parallel access to vector components stored in different memory modules.

•Shared Memory MIMD uses IN to access shared memory.



(a) Bus (b) Fully connected

Figure 1-12
Two possible interconnection networks.

The Evolution of Parallel Computers

SIMD and MIMD Programming

Getting Started

•Parallel Programming Language: How does a parallel algorithm executed by a given parallel processor?

•Generation of pseudocode for a traditional SISD computer similar to the language used.

•Pseudocode needs control, structures, assignment statements and comments that explain each process.

•Use of conventional mathematical operations for relational operations (<,>,= ,etc.).

• Extend serial processor pseudocode (SISD) to be used in vector procesors (SIMD).

The Evolution of Parallel Computers

SIMD and MIMD Programming

Pseudocode Conventions.

Sequential pseudo code will use the convention summarized below:

Assignment:	:=
End of Statement	;
Statement Grouping	begin ... end or { ... }
Statement Label	LABEL:
Counted Loops	for i := 0 step s until m <statement>;
Conditional	if m > p then <statement> else <statement>;
Comments	/*comments*/

SIMD Extensions to Pseudo Code

For a vector assignment statement the format is:

<indexed variable> := <indexed expression> , (<index range>) ;

The Evolution of Parallel Computers

SIMD and MIMD Programming

Pseudocode extensions for describing Multiprocessor algorithms.

Process control and storage class definition:

fork<label> Start a process executing at <label>;

join<integer> join <integer> processes into one;

private <variable list>
Make the storage class private;

If explicit typing of all variables is desired we will include:

shared <variable list>
Make the storage class shared;

The Evolution of Parallel Computers

SIMD and MIMD Programming

Pseudocode example for Matrix Multiplication for SIMD.

```
/* Matrix multiply, C := A*B. Compute elements of C by :          */
/*  C_ij = sum_{K=0}^{N-1} A_ik . B_kj                            */

For i := 0 step 1 until N-1
      begin /* Compute one row of C*/
      /* initialize the sums for each element of a row of C       */

      C[i,j] := 0 ,                (0≤ j ≤ N–1) ;

      /* Loop over the terms of the inner product                 */

         for k := 0 step 1 until N-1

         /* Add the kth inner product term across columns in parallel */

         C[i,j] := C[i,j] + A[i,k] * B[K,j],      (0 ≤ j ≤ N-1) ;

      end /* of all rows */
```

The Evolution of Parallel Computers

SIMD and MIMD Programming

Pseudocode example for Matrix Multiplication for SIMD.

A 3*3 matrix multiply

$$\begin{matrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{matrix} \quad \begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{matrix} \quad \begin{matrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{matrix}$$

Row 0 of C **for I=0, K=0, $0 \le J \le N-1$**

$[a_{00}b_{00}, a_{00}b_{01}, a_{00}b_{02}]$

Row 0 of C **for I=0, K=1, $0 \le J \le N-1$**

$[a_{00}b_{00}+a_{01}b_{10}, a_{00}b_{01}+a_{01}b_{11}, a_{00}b_{02}+a_{01}b_{12}]$

Row 0 of C **for I=0, K=2, $0 \le J \le N-1$**

$[a_{00}b_{00}+a_{01}b_{10}+a_{02}b_{20}, a_{00}b_{01}+a_{01}b_{11}+a_{02}b_{21}, a_{00}b_{02}+a_{01}b_{12}+a_{02}b_{22}]$

After completing the K loop, one row of C is completely generated.

Completion of I loop, completes the matrix multiplication.

The Evolution of Parallel Computers

SIMD and MIMD Programming

Matrix Multiply Pseudocode Example for Multiprocessor. p17

```
/* Matrix multiply, C := A*B. Compute elements of C by :          */
/*  C_ij = sum_{K=0}^{N-1} A_ik . B_kj                            */
                    Private i, j, k

/* Start N-1 new processes, each for a different column of C */

                    For j:= 0 step 1 until N-2
                        fork  DOCOL;

/* The original process reaches this point and does column N-1 */

                    j:= N-1 ;

DOCOL:   /* Executed by N processes, each doing one column of C */
                    for i := 0 step 1 until N-1
                        begin /* compute a row element of C */
                        /* initialize the sum for the inner product */

                        C[i,j] := 0;

                        /*Loop over the terms of the inner product*/

                        for k := 0 step 1 until N-1
                            /* Add the kth inner product term */
                            C[i,j] := C[i,j] + A[i,k] * B[k,j] ;

                        end /* of all rows */

                    join N ;
```

The Evolution of Parallel Computers

SIMD and MIMD Programming

Parallelism in Algorithms and data dependencies. P17

Three types of dependencies: Output, Flow and Anti-dependence

1. $O(S_i) \cap O(S_j) = \Phi$ output independence
2. $I(S_j) \cap O(S_i) = \Phi$ flow independence
3. $I(S_i) \cap O(S_j) = \Phi$ anti independence