

# Introduction to Computer Architecture (Parallel and Pipeline processors)

KR Chowdhary  
Professor & Head  
*Email: kr.chowdhary@gmail.com*  
*webpage: krchowdhary.com*

Department of Computer Science and Engineering  
MBM Engineering College, Jodhpur

November 22, 2013

- ▶ One classification by M.J. Flynn considers the organization of a computer system by the number of *instructions* and *data* items that can be manipulated simultaneously.
  - ▶ The sequence of instructions read from the memory constitute an *instruction stream*.
  - ▶ The operation performed on data in the processor constitutes a *data stream*.
  - ▶ Parallel processing may occur in *instruction stream* stream or *data stream*, or both.
1. **Single-instruction single-data streams (SISD)**: Instructions are executed sequentially.
  2. **Single-instruction multiple-data streams (SIMD)**: All processors receive the same instruction from control unit, but operate in different sets of data.
  3. **Multiple-instruction single-data streams (MISD)**: It is of theoretical interest only as no practical organization can be constructed using this organization.
  4. **Multiple-instruction multiple-data streams (MIMD)**: Several programs can execute at the same time. Most multiprocessors come in this category.

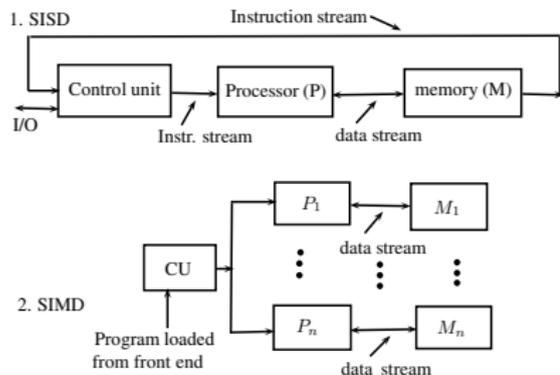


Figure 1: SISD and SIMD architecture

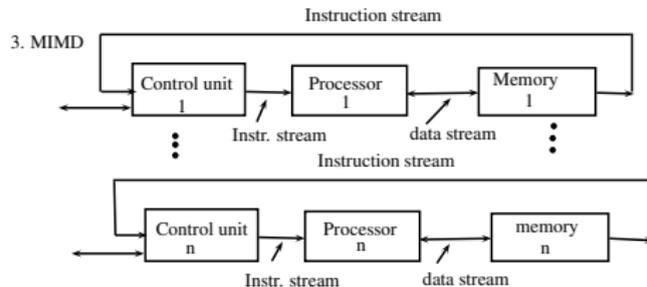


Figure 2: MIMD Architecture.

One of the parallel processing class that does not fit into this classification is *pipeline processing*.

## Parallel Processing:

- ▶ Increasing speed by doing many things in parallel.
- ▶ Let  $P$  is a sequential processor processing the task  $T$  in sequential manner. If  $T$  is partitioned into  $n$  subtasks  $T_1, T_2, \dots, T_n$  of approx. same size, then a processor  $P'$  (say) having  $n$  processors can be programmed so that all the subtasks of  $T$  can execute in parallel.
- ▶ Then  $P'$  executes  $n$  times faster than  $P$ .
- ▶ A failure of CPU is fatal to a sequential processor, but not in the case of parallel processor.
- ▶ Some of the applications of parallel computer (processors) are:
  1. Expert system for AI
  2. Fluid flow analysis,
  3. Seismic data analysis
  4. Long range weather forecasting,
  5. Computer Assisted tomography
  6. Nuclear reactor modeling,
  7. Visual image processing
  8. VLSI design
- ▶ The typical characteristic of parallel computing are: vast amount of computation, floating point arithmetic, vast number of operands.

- ▶ We assume that a given computation can be divided into concurrent tasks for execution on the multiprocessor.
- ▶ Equal Duration Model: A given task can be divided into  $n$  equal subtasks, each of which can be executed by one processor. If  $t_s$  is the execution time of the whole task using a single processor, then the time taken by each processor to execute its subtask is  $t_m = t_s/n$ . Since, according to this model, all processors are executing their subtasks simultaneously, then the time taken to execute the whole task is  $t_m = t_s/n$ .
- ▶ The speedup factor of a parallel system can be defined as the *ratio between the time taken by a single processor to solve a given problem instance to the time taken by a parallel system consisting of  $n$  processors to solve the same problem instance.*

$$S(n) = \text{speedup factor}$$

$$\begin{aligned} &= \frac{t_s}{t_m} \\ &= \frac{t_s}{t_s/n} = n \end{aligned}$$

- ▶ A typical example of parallel processing is a one-dimensional array of processors, where there are  $n$  identical processors  $P_1 \dots P_n$  and each having its local memory. These processors communicate by message passing (send - receive).

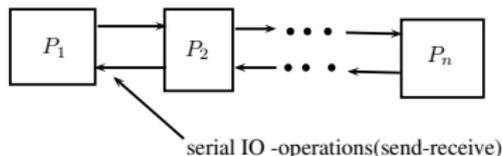


Figure 3: Pipeline processing.

- ▶ There are total  $n$  operations going on in parallel.
- ▶ A pipe line constitutes a sequence of processing circuits, called segments or stages.
- $m$  stage pipeline has same throughput as  $m$  separate units.

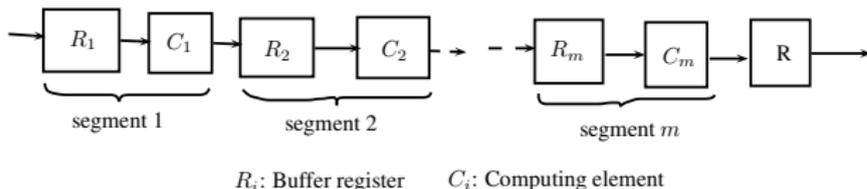


Figure 4: Pipeline segments

1. **Instruction pipeline:** Transfer of instructions through various stages of cpu, during instruction cycle: fetch, decode, execute. Thus, there can be three different instructions in different stages of execution: one getting fetched, previous of that is getting decoded, and previous to that is getting executed.
2. **Arithmetic pipeline:** The data is computed through different stages.

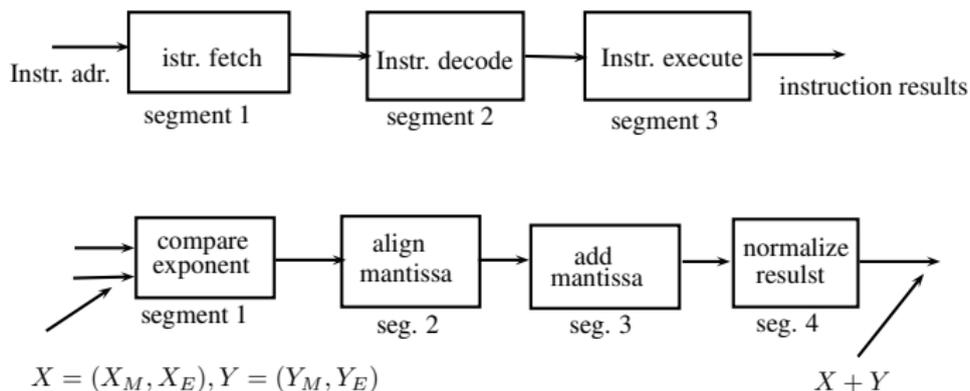


Figure 5: Instruction and data pipeline examples.

- Consider an example to compute:  $A_i * B_i + C_i$ , for  $i = 1, 2, 3, 4, 5$ . Each segment has  $r$  registers, a multiplier, and an adder unit.

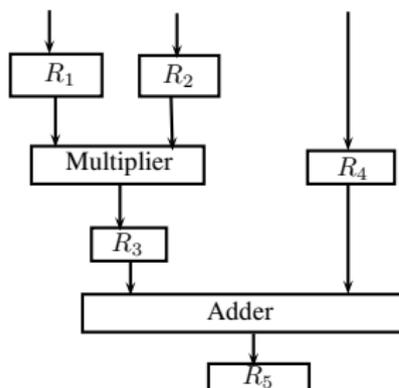


Figure 6: A segment comprising registers and computing elements.

$R_1 \leftarrow A_i, R_2 \leftarrow B_i$ ; input  $A_i, B_i$

$R_3 \leftarrow R_1 * R_2, R_4 \leftarrow C_i$ ; multiply and  $i / C$

$R_5 \leftarrow R_3 + R_4$ ; add  $C_i$  to product

Table 1: Computation of expression  $A_i * B_i + C_i$  in space and time in 3-stage pipeline.

Clock pulse no.	Segment 1 $R_1, R_2$	Segment 2 $R_3, R_4$	Segment 3 $R_5$
1.	$A_1, B_1$	-, -	-
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.	$A_4, B_4$	$A_3 * B_3, C_3$	$A_2 * B_2 + C_2$
5.	$A_5, B_5$	$A_4 * B_4, C_4$	$A_3 * B_3 + C_3$
6.	- -	$A_5 * B_5, C_5$	$A_4 * B_4 + C_4$
7.	- -	-, -	$A_5 * B_5 + C_5$

- ▶ Any operator that can be decomposed into a sequence of sub-operations of about the same components can be implemented by pipeline processor.
- ▶ Consider that for a  $k$ -segment pipeline with clock cycle time  $= t_p$  sec., with total  $n$  no. of tasks ( $T_1, T_2, \dots, T_n$ ) are required to be executed.
- ▶  $T_1$  requires time equal to  $k \cdot t_p$  secs. Remaining  $n - 1$  tasks emerge from the pipeline at the rate of one task per clock cycle, and they will be completed in time of  $(n - 1)t_p$  sec, so total clock cycles required  $= k + (n - 1)$ .
- ▶ For  $k = 3$  segment and  $n = 5$  tasks it is  $3 + (5 - 1) = 7$ , as clear from table 1.

- ▶ Consider an instruction pipeline unit (segment) that performs the same operation and takes time equal to  $t_u$  to complete each task. Total time for  $n$  tasks is  $n.t_u$ . The **speedup** for no. of segments as  $k$  and clock period as  $t_p$  is:

$$S(n) = \frac{n.t_u}{(k + (n-1))t_p} \quad (1)$$

- ▶ For large number of tasks,  $n \gg k-1$ ,  $k+n-1 \approx n$ , so,

$$S(n) = \frac{n.t_u}{n.t_p} \quad (2)$$

$$= \frac{t_u}{t_p} \quad (3)$$

- ▶ Instruction pipelining is similar to use of assembly line in manufacturing plant
- ▶ An instruction's execution is broken in to many steps, which indicates the scope for pipelining
- ▶ pipelining requires registers to store data between stages.

Parallel computation with serial section model:

- ▶ It is assumed that fraction  $f$  of a given task (computation) cannot be divided into concurrent subtasks. The remaining part  $(1 - f)$  is assumed to be dividable. (for example,  $f$  may correspond to data i/p).
- ▶ The time required to execute the task on  $n$  processors is:

$$t_m = f \cdot t_s + (1 - f) \cdot \frac{t_s}{n} \quad (4)$$

- ▶ The speedup is therefore,

$$S(n) = \frac{t_s}{f \cdot t_s + (1 - f) \cdot \frac{t_s}{n}} \quad (5)$$

$$= \frac{n}{1 + (n - 1) \cdot f} \quad (6)$$

- ▶ So,  $S(n)$  is primarily determined by the code section, which cannot be divided.
- ▶ If task is completely serial ( $f = 1$ ), then no speedup can be achieved even by parallel processors.
- ▶ For  $n \rightarrow \infty$ ,

$$S(n) = \frac{1}{f} \quad (7)$$

which is maximum speedup.

- ▶ Improvement in performance (speed) of parallel algorithm over a sequential is limited not by no. of processors but by fraction of the algorithm (code) that cannot be parallelized. (Amdahl's law).
- ▶ Considering the communication overhead:

$$S(n) = \frac{t_s}{f \cdot t_s + (1-f)(t_s/n) + t_c} \quad (8)$$

$$= \frac{n}{f \cdot (n-1) + 1 + n(t_c/t_s)} \quad (9)$$

- ▶ For  $n \rightarrow \infty$ ,

$$S(n) = \frac{n}{f(n-1) + 1 + n(t_c/t_s)} \quad (10)$$

$$= \frac{1}{f + (t_c/t_s)} \quad (11)$$

- ▶ Thus,  $S(n)$  depends on communication overhead  $t_c$  also.

**Instruction Pipe-lining:** typical stages of pipeline are:

1. FI (fetch instruction)
2. DI (decode Instruction)
3. CO (calculate operands)
4. FO (fetch operands)
5. EI (execute instruction)
6. WO (write operands)

- ▶ Nine different instructions are to be executed
- ▶ The six stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.

time units →

Instruc.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
In1	FI	DI	CO	FO	EI	WO								
In2		FI	DI	CO	FO	EI	WO							
In3			FI	DI	CO	FO	EI	WO						
In4				FI	DI	CO	FO	EI	WO					
In5					FI	DI	CO	FO	EI	WO				
In6						FI	DI	CO	FO	EI	WO			
In7							FI	DI	CO	FO	EI	WO		
In8								FI	DI	CO	FO	EI	WO	
In9									FI	DI	CO	FO	EI	WO

- ▶ The diagram assumes that each instruction goes through 6 stages of pipeline.
- ▶ But, for example, a load instruction does not need WO.
- ▶ It is also assumed that there is no memory conflicts, for example, FI, FO, WO in all require memory access (together).
- ▶ The value may be in cache, or FO/WO may be null.
- ▶ Six stages may not be of equal duration, conditional branch/interrupt instruction may invalidate several fetches
- ▶ After which stage it should check for conditional branch/interrupt?

- ▶ Overhead in each stage of pipeline for data movements buffer to buffer
  - ▶ Amount of control logic needed to handle memory/register dependencies increases with size of pipeline
  - ▶ It needs time for the buffers to operate
- ▶ **Pipeline Hazard** occur when pipeline or its portion stalls.
    1. **Resource hazard:** Two or more instructions in pipeline require same resource (say ALU/reg.) (called structure hazard)
    2. **Data hazards:** conflict in memory access
    3. **Control hazards:** (called branch hazards) wrong decision in branch prediction

- ▶ In many computational applications, a problem can be formulated in terms of vectors and matrices. Processing these by a special computer is called **vector processing**.
- ▶ A vector is:  
 $V = [V_1 V_2 V_3 \dots V_n]$ . The index for  $V_i$  is represented as  $V[i]$ . A program for adding two vectors  $A$  and  $B$  of length 100, to produce vector  $C$  is:
- ▶ Scalar Concept:  

```
for(i=0; i < 100; i++)  
  c[i]=b[i]+a[i];
```
- ▶ In machine language we write it as:

```
        mvi i, 0  
loop:  read A[i]  
        read B[i]  
        store i = i +1  
        cmp i, 100  
        jnz loop
```

- ▶ Accesses the arrays  $A$  and  $B$ , and only counter needs to be updated. The vector processing computer eliminates the need of fetching the instructions, and executing them. As they are fetched only once only, decoded once only, but executes them 100 times. This allows operations to be specified only as:

$$C(1:100) = A(1:100) + B(1:100)$$

- ▶ Vector instructions includes the initial address of operands, length of vectors, and operands to be performed, all in one composition instruction. The addition is done with a pipelines floating pointing point adder. It is possible to design vector processor to store all operands in registers in advance.
- ▶ It can be applied in matrix multiplication, for  $[l \times m] \times [m \times n]$ .

► **CISC:** (Complex instruction set computer)

1. Complex programs have motivated the complex and powerful HLL. This produced *semantic gap* between HLL and machine languages, and required more effort in compiler constructions.
2. The attempt to reduce the semantic gap/simplify compiler construction, motivated to make more powerful instruction sets. (CISC - **Complex Instruction set computing**)
3. CISC provide better support for HLLs
4. Lesser count of instructions in

program, thus small size, thus lesser memory, and faster access

► **RISC:** (Reduced instruction set computer)

1. Large number of Gen. purpose registers, use of compiler technology to optimize register usage
2. R-R operations (**Adv.?**)
3. Simple addressing modes
4. Limited and simple instruction set (one instruction per machine cycle)
5. **Advantage of simple instructions?**
6. Optimizing pipeline

1. Design a pipeline configuration to carry out the task to compute:

$$(A_i + B_i)/(C_i + D_i)$$

2. Construct pipeline to add 100 floating point numbers, i.e., find the result of  $x_1 \times x_2 \times \dots \times x_{100}$ .
3.
  - 3.1 List the advantages of designing a floating point processor in the form of a  $k$ -segment pipeline rather than a  $k$ -unit parallel processor.
  - 3.2 A floating-point pipeline has four segments  $S_1, S_2, S_3, S_4$ , whose delays are 100, 90, 100, and 110 nano-secs, respectively. What is the pipeline's maximum throughput in MFLOPS?
4. It is frequently argued that large (super) computer is approaching its performance limits, and the future advances in large computers will depend on interconnected large number of inexpensive computers together. List the arguments against and favor.
5. List the features to be added in sequential programming languages, to use them in large interconnection of small inexpensive computers.
6. Let  $S_1, S_2, \dots, S_k$  denote the sequence of  $k$ -operations on a program. Suppose that execution of these operations on a uni-processor produces the same results regardless of the order of execution of the  $k$   $S_i$ 's. Show that this does not imply that the  $S_i$ 's are parallelizable on a multi-processor.

7. Prove that general problem of determining whether two program segments  $S_1, S_2$  are parallelizable is undecidable by showing that a solution to this problem implies a solution to the halting problem for Turing machine. (Hint: Assume that an algorithm  $A$  to determine parallelization exists, and consider applying  $A$  to a program containing the statement: **if** Turing machine  $T$  halts after at most  $n$  steps **then**  $S_1$  **else**  $S_2$ ).

-  M. Morris Mano, "Computer System Architecture", 3rd Edition, Pearson, 2006.
-  William Stalling, "Computer Organization and Architecture", 8th Edition, Pearson, 2010.
-  John P. Hayes, "Computer Architecture and Organization", 2nd Edition, McGraw-Hill International Edition, 1988.