www.AllAbtEngg.com

UNIT IV

PARALLELISM

Instruction-level-parallelism – Parallel processing challenges – Flynn's classification – Hardware multithreading – Multicore processors

4.1 Instruction-Level parallelism (ILP)

There are two primary methods for increasing the potential amount of instruction-level parallelism.

1. Increasing the depth of the pipeline to overlap more instructions.
2. To replicate the internal components of the computer

Multiple issue:

- It is a technique used to launch multiple instructions in every pipeline stage. Launching multiple instructions per stage allows the instruction execution rate to exceed the clock rate the CPI to be less than one.
- There are two major ways to implement a multiple-issue processor such as
    1. Static multiple issue
    2. Dynamic multiple issue.
- The major differences between these two kinds of issues are the division of work between the compiler and the hardware, because the division of work dictates whether decisions are made at compile time during execution

Static multiple issue:

- It is an approach to implementing a multiple-issue processor where many decisions are made by the compiler before execution.

Dynamic multiple issue :

- It is an approach to implementing a multiple-issue processor where many decisions are made during execution by the processor.

Multiple-issue pipeline:

- There are two primary and distinct responsibilities that must be dealt with in a multiple-issue pipeline:
    1. Packaging instructions into issue slots
    2. Dealing with data and control hazards.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

Packaging instructions into issue slots:

- Issue slots are positions from which instructions could issue in a given clock cycle; by analogy these correspond to positions at the starting blocks for a sprint.
- How does the processor determine how many instructions and which instructions can be issued in a given clock cycle?
- In most static issue processors, this process is at least partially handled by the compiler and in case of dynamic issue processor it is dealt with runtime by the processor.

Dealing with data and control hazards:

- In static issue processors, some or all of the data and control hazards are handled statically by the compiler.
- In dynamic issue processors at least some classes of hazards using hardware techniques operating at execution time.

4.1.1 The Concept of Speculation

- One of the most important methods for finding and exploiting more ILP is speculation.
- Speculation is an approach that allows the compiler or the processor to guess about the properties of an instruction, so as to enable execution to begin for other instructions that may depend on the speculated instruction.
- Speculation is an approach whereby the compiler or processor guesses the outcome of an instruction to remove it as dependence in executing other instructions.
- For example, we might speculate on the outcome of a branch, so that instructions after the branch could be executed earlier.
- Speculation may be done in the compiler or by the hardware. For example, the compiler can use speculation to reorder instructions, moving an instruction across a branch or a load across a store.

Difficulty with speculation:

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- The difficulty with speculation is that it may be wrong. So, any speculation mechanism must include both a method to check if the guess was right and a method to unroll or back out the effects of the instructions was executed speculatively. But it will add complexity.
- The recovery mechanisms used for incorrect speculation is done in two ways:
- Hardware speculation
- Software speculation
- In hardware speculation the processor usually buffers the speculative results until it knows they are no longer speculative.
- If the speculation was correct, the instructions are completed by allowing the contents of the buffers to be written to the registers or memory.
- If the speculation was incorrect, the hardware flushes the buffers and re-executes the correct instruction sequence.
- In software speculation the compiler usually inserts additional instructions that check the accuracy of the speculation and provide a fix-up routine to use when the speculation was incorrect.
- Speculation can improve performance when done properly and decrease performance when it is done carelessly.

4.1.2 Static Multiple Issue

- Static multiple-issue processors use the compiler to assist with packaging instructions and handling hazards. It will use the issue packet.
- Issue packet is a set of instructions that issue together in one clock cycle and the packet may be determined statically by the compiler or dynamically by the processor
- Static multiple-issue processor usually restricts what mix of instructions can be initiated in a given clock cycle, it is useful to think of the issue packet as a single instruction allowing several operations in certain predefined fields. This view led to the original name for this approach: Very Long Instruction Word (VLIW).

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- VLIW is a style if instruction set architecture that launches many operations that are defined to be independent in a single wide instruction and it has many separate opcode fields.
- Most static issue processors compiler take some responsibility for handling data and control hazards. The compiler's responsibilities may include
    1. Static branch prediction
    2. Code scheduling to reduce or prevent all hazards.

An Example: Static Multiple Issue with the MIPS ISA:

- Static multiple issue processor functions can be explained using two-issue MIPS processor.The two-issue MIPS processor one of the instructions can be an integer ALU operation or branch and the other can be a load or store.
- These kinds of designs are in embedded MIPS processors. Issuing two instructions per cycle will require fetching and decoding 64 bits of instructions.
- In many static multiple-issue processors and all VLIW processors and all VLIW processors, issuing instructions simultaneously is restricted to simplify the decoding and instruction issue.

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

Figure Static two-issue pipeline in operation.

- So instructions must be paired and aligned on a 64-bit boundary with the ALU or branch portion must appear first.
- If one instruction of the pair cannot be used that is replaced with a nop. Figure shows static two-issue pipeline in operation.
- The ALU and data transfer instructions are issued at the same time. Here we have assumed the same five-stage structure as used for the single-issue pipeline.

- Keeping the register writes at the end of the pipeline will simplifies the handling of exceptions and the maintenance of a precise exception model. But it is more difficult in multiple-issue processors.
- In some static multiple issue processors the compiler takes full responsibility for removing all hazards, scheduling the code and inserting no-ops. For these kinds of design the code executes without any need for hazard detection or hardware-generated stalls.

- The hardware can detect data hazards and generates stalls between two issue packets for that the compiler avoid all dependences within an instruction pair.
- The software must handle all hazards or only try to reduce the fraction of hazards between separate issue packets.
- To perform ALU and a data transfer operation in parallel, first we need for additional hardware.
- With additional hardware units some other units also necessary such as hazard detection and stall logic.
- In 1 clock cycle we need to perform the following task:
    1. read two registers for the ALU operation
    2. two more for a store
    3. one register for write port for a load
    4. one write port for an ALU operation
- ALU is tied with ALU operation, so need a separate adder to calculate the effective address.This process is shown in below figure.
- Two-issue processor will improve performance .But many instructions be overlapped in execution, and additional overlap increases the relative performance loss from data and control hazards.
- In our simple five-stage pipeline loads have a use latency of 1 clock cycle and it which prevents one instruction from using the result without stalling.
- Use latency is a number of clock cycle between a load instruction and an instruction can use the results of the load without stalling the pipeline.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

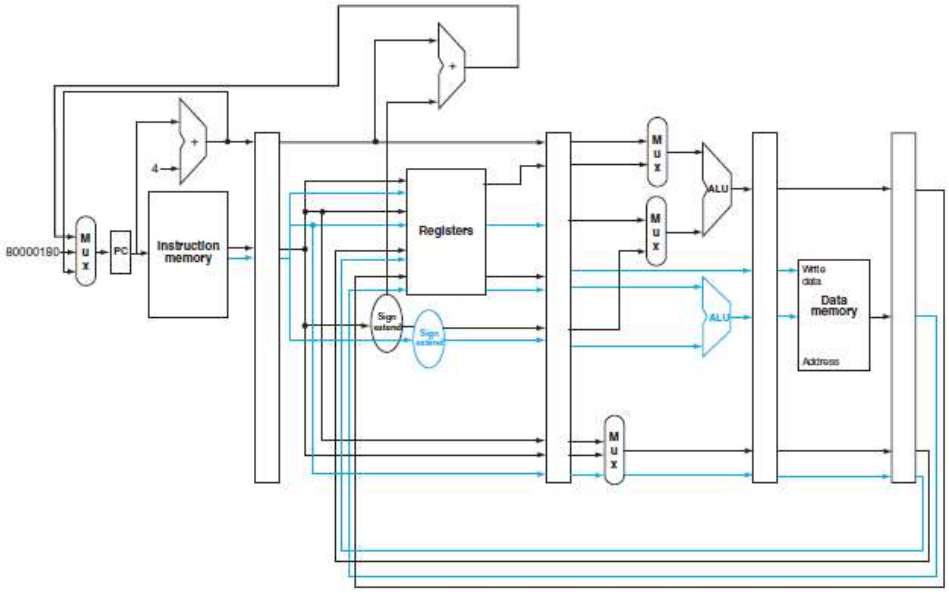

Figure: A static two-issue datapath

- In the two-issue a load instruction cannot be used on the next clock cycle it means that the next two instructions cannot use the load result without stalling.
- In a multiple-issue processor more determined compiler or hardware scheduling techniques are needed. But in static multiple issue it can be done by compiler itself.

Simple Multiple-Issue Code Scheduling:

Example:1

How would this loop be scheduled on a static two-issue pipeline for MIPS?

```
Loop: lw $t0, 0($s1) # $t0=array element
      addu $t0,$t0,$s2 # add scalar in $s2
      sw $t0, 0($s1) # store result
      addi $s1,$s1,–4 # decrement pointer
      bne $s1,$zero,Loop # branch $s1!=0
```

Reorder the instructions to avoid as many pipeline stalls as possible. Assumebranches are predicted, so that control hazards are handled by the hardware.

Answer:

The first three instructions have data dependences, and so do the last two instruction and for that do schedule.

	ALU or branch instruction	Data transfer instruction	Clock cycle
Loop:		lw $t0, 0($s1)	1
	addi $s1,$s1,-4		2
	addu $t0,$t0,$s2		3
	bne $s1,$zero,Loop	sw $t0, 4($s1)	4

Figure: The scheduled code as it would look on a two-issue MIPS pipeline.

- The empty slots are nops.
- It takes 4 clocks per loop iteration; at 4 clocks to execute 5 instructions

Loop unrolling:

- Loop unrolling is an compiler technique to get more performance from where multiple copies of the loop body are made.
- It is a technique to get more performance from loops that access arrays, in which multiple copies of the loop body are made and instructions from different iterations .

Loop Unrolling for Multiple-Issue Pipelines:

Example:2

Let's consider the above example and know how Loop Unrolling and scheduling work for simplicity ,Assume that the loop index is a multiple of four.

Answer:

To schedule the loop without any delays, it turns out that we need to make four copies of the loop body. After unrolling and eliminating the unnecessary loop overhead instructions, the loop will contain four copies each of lw, add, and sw, plus one addi and one bne.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

	ALU or branch instruction		Data transfer instruction		Clock cycle
Loop:	addi	$s1,$s1,-16	lw	$t0, 0($s1)	1
			lw	$t1,12($s1)	2
	addu	$t0,$t0,$s2	lw	$t2, 8($s1)	3
	addu	$t1,$t1,$s2	lw	$t3, 4($s1)	4
	addu	$t2,$t2,$s2	sw	$t0, 16($s1)	5
	addu	$t3,$t3,$s2	sw	$t1,12($s1)	6
			sw	$t2, 8($s1)	7
	bne	$s1,$zero,Loop	sw	$t3, 4($s1)	8

Figure: The unrolled and scheduled code for previous problem.

- The empty slots are nops. Since the first instruction in the loop decrements $s1 by 16, the addresses loaded are the original value of $s1, then that address minus 4, minus 8, and minus 12.
- During the unrolling process, the compiler introduced additional registers ($t1, $t2, $t3). The goal of this process, called registers renaming.

Register renaming:

- The renaming of registers, by the compiler or hardware to remove ant dependences.

Ant dependence:

- It is also called name dependence. Which is an ordering forced by the reuse of a name rather than a real data dependence that is also called a true dependence.
- Renaming the registers during the unrolling process will allows the compiler to move these independent instructions subsequently, so we obtain better schedule the code. The renaming process eliminates the name dependences, while preserving the true dependences.

4.1.3 Dynamic Multiple-Issue Processors

- Dynamic multiple-issue processors are also known as **superscalar** processors, or simply superscalar.
- An advanced pipelining technique that enables the processor to execute more than one instruction per clock cycle.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- Achieving good performance on such a processor still requires the compiler to try to schedule instructions to move dependences apart and thereby improve the instruction issue rate.
- Even with such compiler scheduling, there is an important difference between this simple superscalar and a VLIW processor.
- In superscalar processor whether the code has scheduled or not, is guaranteed by the hardware to execute correctly.
- In superscalar processor the compiled code can run correctly independent of the issue rate or pipeline structure of the processor.
- In some VLIW designs, this has not been the case, and it require recompilation when moving across different processor models.
- In some static issue processors the code can run correctly across different implementations, but it requires poorly compilation.
- Many superscalar extend the basic framework of dynamic issue decisions to include dynamic pipeline scheduling.
- Dynamic pipeline scheduling chooses which instructions to execute in a given clock cycle while trying to avoid hazards and stalls.
- Let's consider the simple example to avoid a data hazard. Consider the following code sequence:

```
lw $t0, 20($s2)
addu $t1, $t0, $t2
sub $s4, $s4, $t3
slti $t5, $s4, 20
```

- Even though the sub instruction is ready to execute, it must wait for the lw and addu to complete first, which might take many clock cycles if memory is slow. Dynamic pipeline scheduling allows such hazards to be avoided either fully or partially.

Dynamic Pipeline Scheduling:

- Dynamic pipeline scheduling chooses which instructions to execute next, possibly reordering them to avoid stalls. In such processors, the pipeline is divided into three major units:

1. An instruction fetch and issue unit
2. Multiple functional units
3. Commit unit.

- Figure shows the model. The final step of updating the state is also called retirement or graduation.

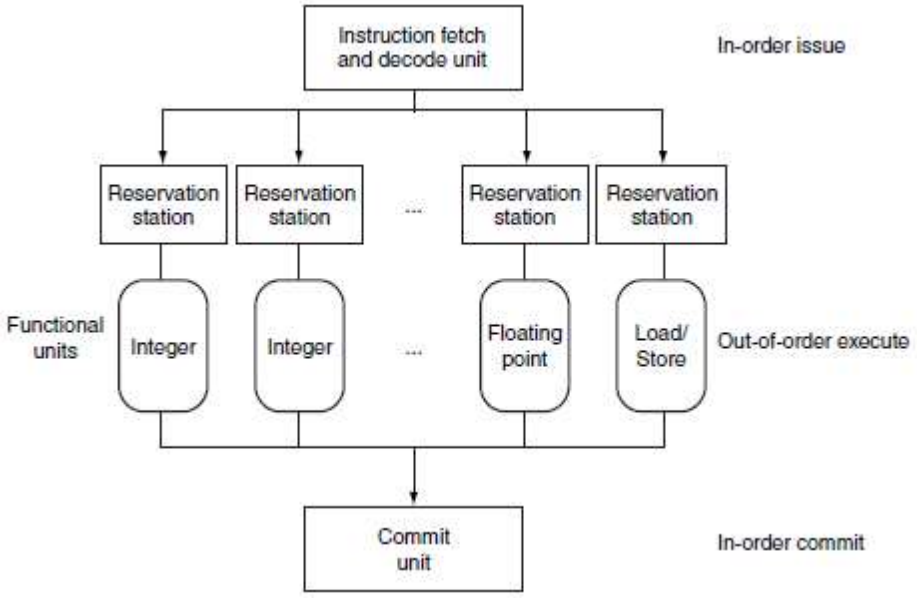

Figure: The three primary units of a dynamically scheduled pipeline

- The first unit fetches instructions, decodes them, and sends each instruction to a corresponding functional unit for execution.
- Each functional unit has buffers, called **reservation stations** that hold the operands and the operation.
- The buffer contains all its operands and the functional unit is ready to execute, the result is calculated.
- When the result is completed it is sent to any reservation stations waiting for this particular result as well
- as to the commit unit.
- The result must be buffered until it is safe to put the result into the register file or, for a store, into memory.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- The buffer in the commit unit often called the **reorder buffer**. The buffer that holds results in a dynamically scheduled processor until it is safe to store the results to memory
- Once a result is committed to the register file, it can be fetched directly from there, just as in a normal
- pipeline.
- The form of register renaming is obtained by combination of buffering operands in the reservation stations and results in the reorder buffer.
- Below two steps are explained how the renaming of registers is obtained.

Step:1

- When an instruction issues it is copied to the reservation station for the appropriate functional unit.
- If any operands are available in the register file or the reorder buffer are also copied to the reservation station immediately.
- The instruction is buffered in the reservation station until all the operands and an functional unit are available.
- For the issuing instruction, the register copy of the operand is no longer required, and if a write to that register occurred, the value could be overwritten.

Step:2

- If an operand is not in the register file or reorder buffer, it must be waiting to be produced by a functional unit.
- The name of the functional unit that will produce the result is tracked. When that unit eventually produces the result, it is copied directly into the waiting reservation station bypassing the registers from the functional unit
- These two steps use the reorder buffer and the reservation stations to implement register renaming.

In-order commit:

- A commit in which the results of pipelined execution are written to the programmer visible state in the same order that instructions are fetched.

Out-of-order execution:

- A situation in pipelined execution when an instruction blocked from executing does not cause the following instructions to wait.
- Dynamic scheduling is often extended by including hardware-based speculation for branch outcomes. By predicting the direction of a branch, a dynamically scheduled processor can continue to fetch and execute instructions along the predicted path.
- A speculative, dynamically scheduled pipeline can also support speculation on load addresses, allowing load-store reordering, and using the commit unit to avoid incorrect speculation.

4.2 Parallel Processing Challenges

Parallel processing:

- It is a single program that runs on multiple processors simultaneously.
- It is also called as data-level parallelism.
- Parallel processing programs that will execute efficiently in performance and power as the number of cores per chip. Parallel processing program or parallel software means either sequential or concurrent software running on parallel hardware.
- The difficulty with parallelism is not the hardware; it is that too few important application programs have been rewritten to complete tasks sooner on multiprocessors. It is difficult to write software that uses multiple processors to complete one task faster, and the problem gets worse as the number of processors increases.

Why have parallel processing programs been so much harder to develop than sequential programs?

- The first reason is that you *must* get better performance and efficiency from a parallel processing program on a multiprocessor; otherwise, you would just use a sequential program on a uni-processor, as programming is easier.
- In fact, uni-processor design techniques such as superscalar and out-of-order execution take advantage of instruction-level parallelism normally without the involvement of the programmer.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- Such improvement reduced the demand for rewriting programs for multiprocessors, since programmers could do nothing and yet their sequential programs would run faster on new computers.

Why is it difficult to write parallel processing programs that are fast, especially as the number of processors increases?

- We used the analogy of eight reporters trying to write a single story in hope of doing the work eight times faster.
- To succeed, the task must be broken into eight equal-sized pieces, because otherwise some reporters would be idle while waiting for the ones with larger pieces to finish.
- Another performance danger would be that the reporters would spend too much time communicating with each other instead of writing their pieces of the story.

- For both this analogy and parallel programming, the challenges include
    1. scheduling,
    2. load balancing,
    3. time for synchronization
    4. overhead for communication between the parties.
- The challenge is hard with the more reporters for a newspaper story and the more processors for parallel programming.

Speed-up Challenge:

Example:1

Suppose you want to achieve a speed-up of 90 times faster with 100 processors. What percentage of the original computation can be sequential?

Solution:

Amdahl's law says

Execution time after improvement =

$$\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

We can reformulate Amdahl's in terms of speed-up versus the original execution time:

$$\text{Speed-up} = \frac{\text{Execution time before}}{(\text{Execution time before} - \text{Execution time affected}) + \frac{\text{Execution time affected}}{100}}$$

This formula is usually rewritten assuming that the execution time before is 1 for some unit of time, and the execution time affected by improvement is considered the fraction of the original execution time:

$$\text{Speed-up} = \frac{1}{(1 - \text{Fraction time affected}) + \frac{\text{Fraction time affected}}{100}}$$

Substituting for the goal of a speed-up of 90 into the formula above:

$$90 = \frac{1}{(1 - \text{Fraction time affected}) + \frac{\text{Fraction time affected}}{100}}$$

Then simplifying the formula and solving for fraction time affected:

$$90 \times (1 - 0.99 \times \text{Fraction time affected}) = 1$$

$$90 - (90 \times 0.99 \times \text{Fraction time affected}) = 1$$

$$90 - 1 = 90 \times 0.99 \times \text{Fraction time affected}$$

$$\text{Fraction time affected} = 89/89.1 = 0.999$$

Thus, to achieve a speed-up of 90 from 100 processors, the sequential percentage can only be 0.1%.

Speed-up Challenge: Bigger Problem:

Example:2

Suppose you want to perform two sums: one is a sum of 10 scalar variables, and one is a matrix sum of a pair of two-dimensional arrays, with dimensions 10 by 10. What speed-up do you get with 10 versus 100 processors? Next, calculate the speed-ups assuming the matrices grow to 100 by 100.

Solution:

If we assume performance is a function of the time for an addition, *t*, then there are 10 additions that do not benefit from parallel processors and 100 additions that do.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

If the time for a single processor is 110 t, the execution time for 10 processors is

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

$$\text{Execution time affected improvement} = \frac{100t}{10} + 10t = 20t$$

so the speed-up with 10 processors is $110t/20t = 5.5$. The execution time for 100 processors is

$$\text{Execution time after improvement} = \frac{100t}{100} + 10t = 11t$$

so the speed-up with 100 processors is $110t/11t = 10$.

Thus, for this problem size, we get about 55% of the potential speed-up with 10 processors, but only 10% with 100. Look what happens when we increase the matrix. The sequential program now takes $10t + 10{,}000t = 10{,}010t$. The execution time for 10 processors is

$$\text{Execution time after improvement} = \frac{10{,}000t}{10} + 10t = 1010t$$

so the speed-up with 10 processors is $10{,}010t/1010t = 9.9$. The execution time for 100 processors is

$$\text{Execution time after improvement} = \frac{10{,}000t}{100} + 10t = 110t$$

so the speed-up with 100 processors is $10{,}010t/110t = 91$. Thus, for this larger problem size, we get about 99% of the potential speed-up with 10 processors and more than 90% with 100.This allows us to introduce two terms that describe ways to scale up.

Strong scaling:

- Strong scaling means measuring speed-up while keeping the problem size fixed.
- Speed-up achieved on a multiprocessor without increasing the size of the problem.

Weak scaling:

- Weak scaling means that the program size grows proportionally to the increase in the number of processors.
- Speed-up achieved on a multiprocessor while increasing the size of the problem proportionally to the increase in the number of processors.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- Let's assume that the size of the problem, M, is the working set in main memory, and we have P processors.
- Then the memory per processor for strong scaling is approximately M/P, and for weak scaling, it is approximately M.

Speed-up Challenge: Balancing Load:

Example:3

To achieve the speed-up of 91 on the previous larger problem with 100 processors, we assumed the load was perfectly balanced. That is, each of the 100 processors had 1% of the work to do. Instead, show the impact on speed-up if one processor's load is higher than all the rest. Calculate at 2% and 5%.

Answer:

If one processor has 2% of the parallel load, then it must do 2% × 10,000 or 200 additions, and the other 99 will share the remaining 9800. Since they are operating simultaneously, we can just calculate the execution time as a maximum

$$\text{Execution time after improvement} = \text{Max}\left(\frac{9800t}{99}, \frac{200t}{1}\right) + 10t = 210t$$

The speed-up drops to $10{,}010t/210t = 48$. If one processor has 5% of the load, it must perform 500 additions:

$$\text{Execution time after improvement} = \text{Max}\left(\frac{9500t}{99}, \frac{500t}{1}\right) + 10t = 510t$$

The speed-up drops even further to $10{,}010t/510t = 20$. This example demonstrates the value of balancing load, for just a single processor with twice the load of the others cuts speed-up almost in half, and five times the load on one processor reduces the speed-up by almost a factor of five.

4.3 Flynn's Classification

- Flynn's Taxonomy distinguishes multi-processor computer architectures according two independent dimensions of Instruction stream and Data stream.
- An instruction stream is sequence of instructions executed by machine.
- And a data stream is a sequence of data including input, partial or temporary results used by instruction stream.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

www.AllAbtEngg.com

- Each of these dimensions can have only one of two possible states: Single or Multiple.

- Flynn's classification depends on the difference between the performance of control unit and the data processing unit rather than its operational and structural interconnections.
- The four categories of Flynn classification and characteristic feature of each of them.
    1. Single instruction stream, single data stream (SISD)
    2. Single instruction stream, multiple data stream (SIMD) processors
    3. Multiple instruction stream, single data stream (MISD)
    4. Multiple instruction stream, multiple data stream (MIMD)

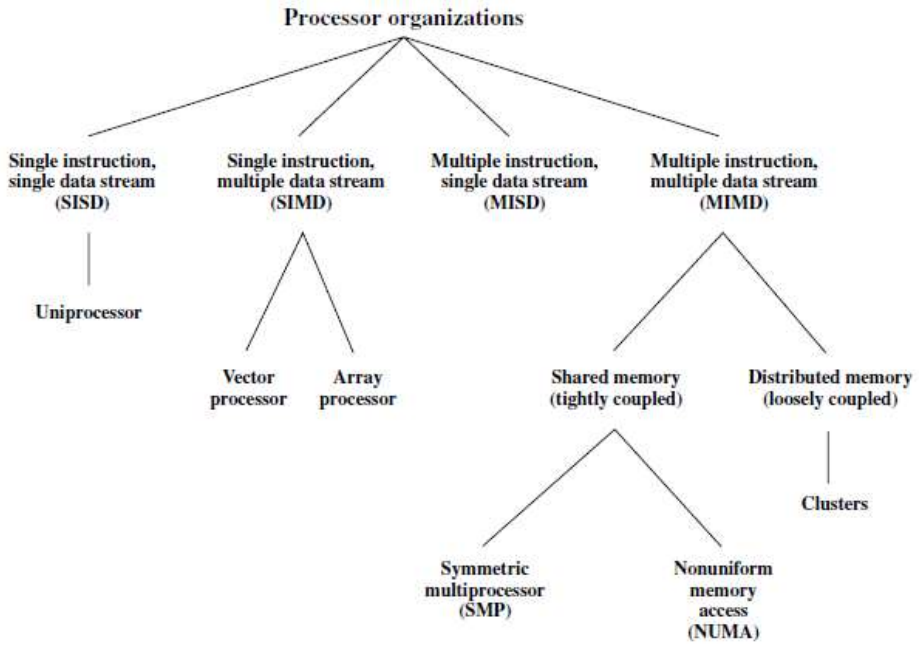

Figure: Taxonomy of Parallel Processor Architectures

4.3.1 Single instruction stream, single data stream (SISD)

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

www.AllAbtEngg.com

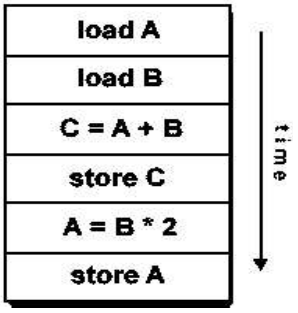

Figure : Execution of instruction in SISD processors

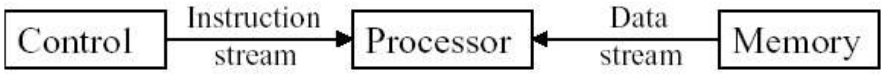

Figure 1.2 SISD processor organization

- The figure organization of SISD computer having one control unit, one processor unit and single memory unit.
- They are also called scalar processor i.e., one instruction at a time and each instruction have only one set of operands.
- Single instruction means only one instruction stream is being acted on by the CPU during any one clock cycle.
- Single data means only one data stream is being used as input during any one clock cycle. Deterministic execution.
- Instructions are executed sequentially.
- This is the oldest and until recently, the most prevalent form of computer.
- Examples: most PCs, single CPU workstations and mainframes

4.3.2 Single instruction stream, multiple data stream (SIMD) processors

- It is a type of parallel computer.
- Single instruction: All processing units execute the same instruction issued by the control unit at any given clock cycle , where there are multiple processor executing instruction given by one control unit.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- Multiple data: Each processing unit can operate on a different data element as shown if figure below the processor are connected to shared memory or interconnection network providing multiple data to processing unit

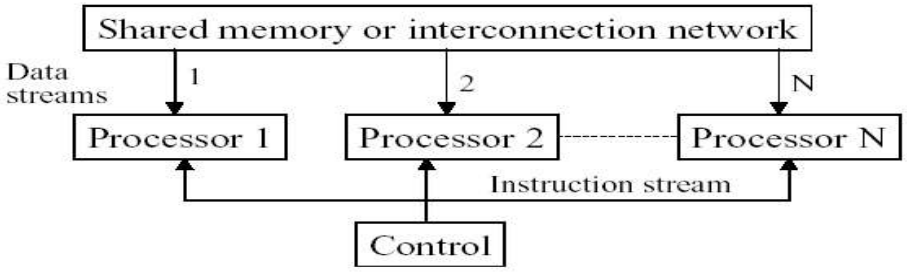

Figure: SIMD processor organization

- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Thus single instruction is executed by different processing unit on different set of data as shown in **SIMD** processor organization.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing and vector computation.

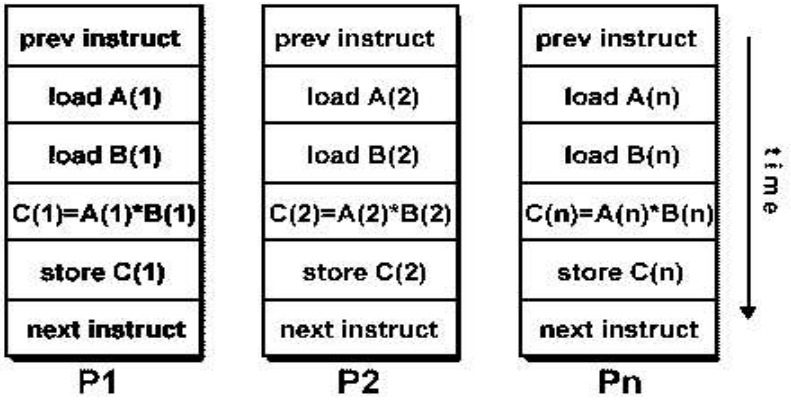

Figure 1.4 Execution of instructions in SIMD processors

- Synchronous (lockstep) and deterministic execution .

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

www.AllAbtEngg.com

- Two varieties: Processor Arrays e.g., Connection Machine CM-2, Maspar MP-1, MP-2 and Vector Pipelines processor e.g., IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

SIMD is categorized as

1. Vector processor
2. Array processor

SIMD Vector processor:

- A vector operand contains an ordered set of n elements, where n is called the length of the vector.
- Each element in a vector is a scalar quantity, which may be a floating point number, an integer, a logical value or a character.
- A vector processor consists of a scalar processor and a vector unit, which could be thought of as an independent functional unit capable of efficient vector operations.

Vector Hardware

- Vector computers have hardware to perform the vector operations efficiently.
- Operands can not be used directly from memory but rather are loaded into registers and are put back in registers after the operation.
- Vector hardware has the special ability to overlap or pipeline operand processing.

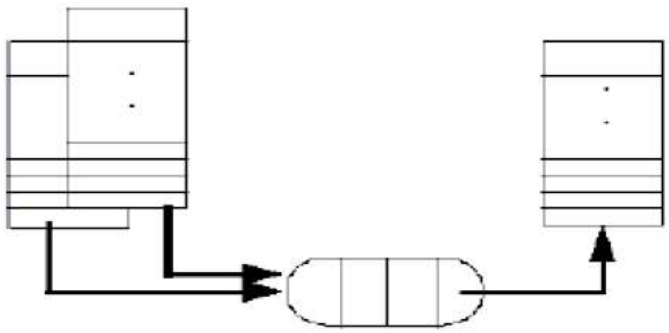

Figure 1.12 Vector Hardware

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- Vector functional units pipelined, fully segmented each stage of the pipeline performs a step of the function on different operand(s) once pipeline is full, a new result is produced each clock period (cp).

Pipelining:

- The pipeline is divided up into individual segments, each of which is completely independent and involves no hardware sharing.
- This means that the machine can be working on separate operands at the same time.
- This ability enables it to produce one result per clock period as soon as the pipeline is full.
- The same instruction is obeyed repeatedly using the pipeline technique so the vector processor processes all the elements of a vector in exactly the same way.
- The pipeline segments arithmetic operation such as floating point multiply into stages passing the output of one stage to the next stage as input.
- The next pair of operands may enter the pipeline after the first stage has processed the previous pair of operands.
- The processing of a number of operands may be carried out simultaneously.
- The loading of a vector register is itself a pipelined operation, with the ability to load one element each clock period after some initial startup overhead.

SIMD Array Processor:

- The Synchronous parallel architectures coordinate Concurrent operations in lockstep through global clocks, central control units, or vector unit controllers.
- A synchronous array of parallel processors is called an array processor.
- These processors are composed of N identical processing elements (PES) under the supervision of a one control unit (CU)
- This Control unit is a computer with high speed registers,local memory and arithmetic logic unit.
- An array processor is basically a single instruction and multiple data (SIMD) computers.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- There are N data streams; one per processor, so different data can be used in each processor.
- SIMD has two basic architectural organizations
    1. Array processor using random access memory
    2. Associative processors using content addressable memory.
- All N identical processors operate under the control of a single instruction stream issued by a central control unit.
- Each PE_i is essentially an arithmetic logic unit (ALU) with attached working registers and local memory PEM_i for the storage of distributed data.
- The CU also has its own main memory for the storage of program.
- The function of CU is to decode the instructions and determine where the decoded instruction should be executed.
- The PE perform same function (same instruction) synchronously in a lock step fashion under command of CU. In order to maintain synchronous operations a global clock is used.
- Thus at each step i.e., when global clock pulse changes all processors execute the same instruction, each on a different data (single instruction multiple data).
- Lets as consider the addition of two vectors each having N element and there are N/2 processing elements in the SIMD.
- The same addition instruction is issued to all N/2 processors and all processor elements will execute the instructions simultaneously.
- It takes 2 steps to add two vectors as compared to N steps on a SISD machine.
- The distributed data can be loaded into PEMs from an external source via the system bus or via system broadcast mode using the control bus. SIMD machines are particularly useful at in solving problems involved with vector calculations where one can easily exploit data parallelism.
- In such calculations the same set of instruction is applied to all subsets of data.
- The array processor can be classified into two category depending on how memory units is organized.
    1. Dedicated memory organization
    2. Global memory organization

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

4.3.3 Multiple instruction stream, single data stream (MISD)

- A single data stream is fed into multiple processing units.Each processing unit operates on the data independently via independent instruction streams as shown in figure MISD processor organization .
- A single data stream is forwarded to different processing unit which are connected to different control unit and execute instruction given to it by control unit to which it is attached.

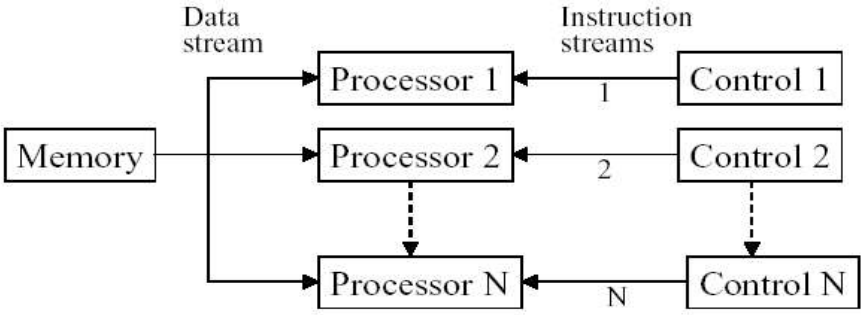

Figure : MISD processor organization

- Thus in these computers same data flow through a linear array of processors executing different instruction streams. This architecture is also known as systolic arrays for pipelined execution of specific instructions.
- Some possible uses might be:
    1. multiple frequency filters operating on a single signal stream
    2. multiple cryptography algorithms attempting to crack a single coded message.

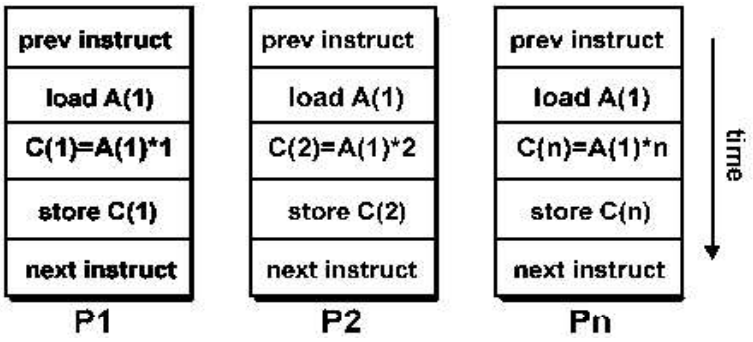

Figure : Execution of instructions in MISD processors

4.3.4 Multiple instruction stream, multiple data stream (MIMD)

- Multiple Instruction: Every processor may be executing a different instruction stream.
- Multiple Data: Every processor may be working with a different data stream, multiple data stream is provided by shared memory.
- It can be categorized as loosely coupled or tightly coupled depending on sharing of data and control.
- Execution can be synchronous or asynchronous, deterministic or non-deterministic

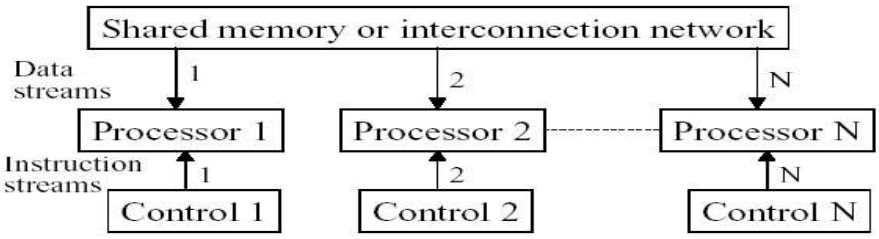

Figure : MIMD processor organizations

- There is different processor each processing different task. Examples: most current supercomputers, networked parallel computer "grids" and multi-processor, SMP computers - including some types of PCs.

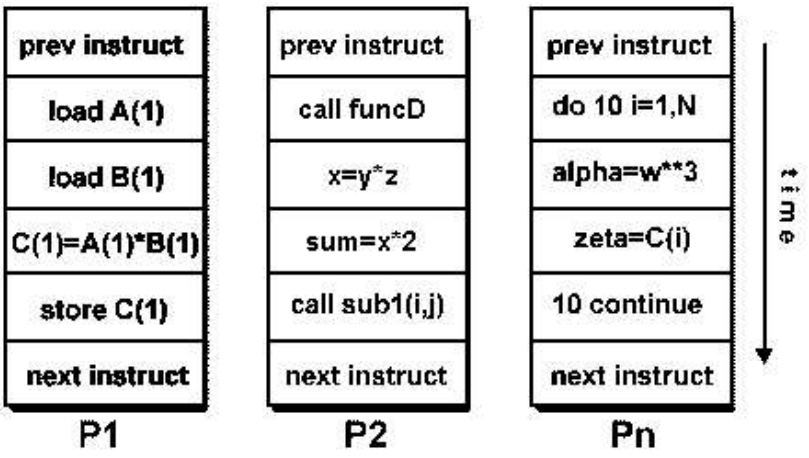

Figure: Execution of instructions MIMD processors

Shared memory multiprocessors:

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources. Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: UMA , NUMA and COMA.

Uniform Memory Access (UMA):

- Most commonly represented by Symmetric Multiprocessor (SMP) machines identical processors.
- Equal access and access times to memory. Sometimes called CC-UMA - Cache Coherent UMA.
- Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update.
- Cache coherency is accomplished at the hardware level.

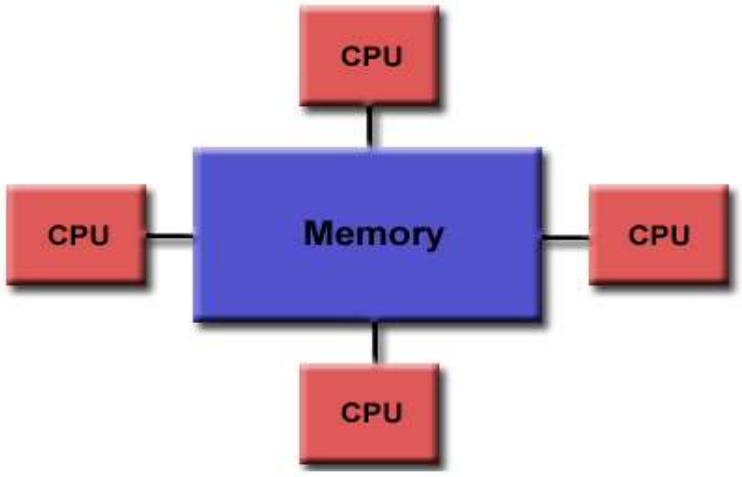

Figure 1.9 Shared Memories (UMA)

Non-Uniform Memory Access (NUMA):

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- Not all processors have equal access time to all memories
- Memory access across link is slower
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA.

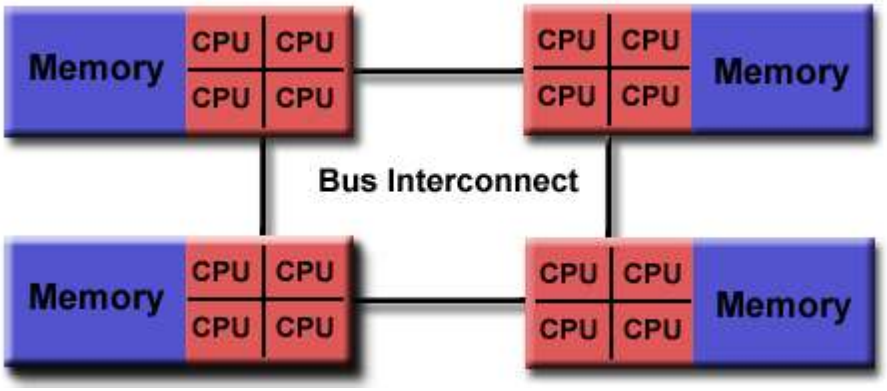

Figure : Shared Memory (NUMA)

The COMA model:

- The COMA model is a special case of NUMA machine in which the distributed main memories are converted to caches.
- All caches form a global address space and there is no memory hierarchy at each processor node.

Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-
- CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.

- Programmer responsibility for synchronization constructs that cover "correct" access of global memory.
- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

- Like shared memory systems, distributed memory systems vary widely but share a common characteristic.
- Distributed memory systems require a communication network to connect inter-processor memory.

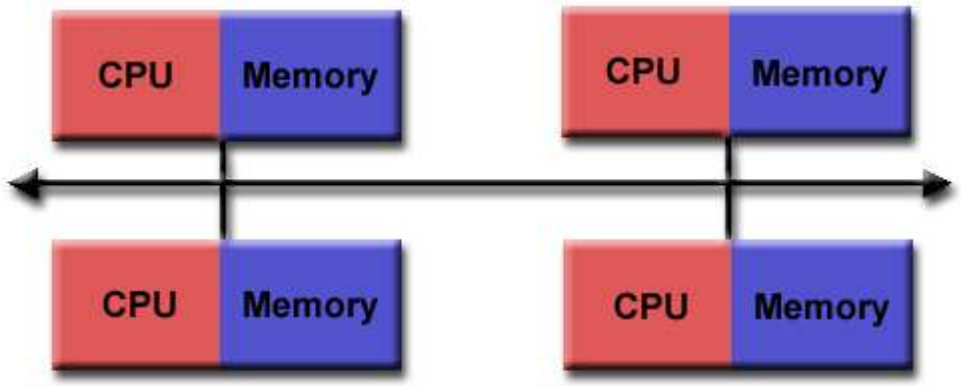

Figure: distributed memory systems

- Processors have their own local memory.
- Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently.
- Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated.
- Synchronization between tasks is likewise the programmer's responsibility.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- Modern multicomputer use hardware routers to pass message. Based on the interconnection and routers and channel used the multicomputers are divided into generation
    1. 1st generation : based on board technology using hypercube architecture and software controlled message switching.
    2. 2nd Generation: implemented with mesh connected architecture, hardware message routing and software environment for medium distributed –grained computing.
    3. 3rd Generation : fine grained multicomputer like MIT J-Machine.
- The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.

Advantages:

- Memory is scalable with number of processor.
- Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness.

Disadvantages:

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access (NUMA) times.

4.4 Hardware Multithreading

- Multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- To permit this sharing, the processor must duplicate the independent state of each thread.
- For example, a separate copy of the register file, a separate PC, and a separate page table are required for each thread.
- The memory itself can be shared through the virtual memory mechanisms, which already support multiprogramming.
- Hardware multithreading increase the utilization of a processor by switching to another thread when one thread is stalled. Hardware must support the ability to change to a different thread relatively quickly.
- Thread is a light weight process and threads share a single address space but process does not share. Thread switch is more efficient than a process switch.
- Process includes one or more threads, the address space and the operating system space. Process switch invokes the operating system but not a thread switch.
- There are two main approaches to multithreading.
    1. Fine-grained multithreading
    2. Coarse-grained multithreading

4.4.1 Fine-grained multithreading

- Fine-grained multithreading switches between threads on each instruction, causing the execution of multiple threads to be interleaved.
- This interleaving is often done in a round-robin fashion, skipping any threads that are stalled at that time. To make fine-grained multithreading practical, the CPU must be able to switch threads on every clock cycle.
- In the fine-grained case, the interleaving of threads eliminates fully empty slots. Because only one thread issues instructions in a given clock cycle,
- ILP limitations still lead to number of idle slots within individual clock cycles.

Advantage:

- It can hide the throughput losses that arise from both short and long stalls, since instructions from other threads can be executed when one thread stalls.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

Disadvantage:

- It slows down the execution of the individual threads, since a thread that is ready to execute without stalls will be delayed by instructions from other threads.

4.4.2 Coarse-grained multithreading

- Coarse-grained multithreading switches threads only on costly stalls, such as level 2 cache misses.
- This change need to have threads switching must be fast and is much less likely to slow down the execution of an individual thread. Because instructions from other threads will only be issued when a thread encounters a costly stall.
- In the coarse-grained multithreaded superscalar, the long stalls are partially hidden by switching to another thread that uses the resources of the processor.
- It will reduce the number of completely idle clock cycles, the ILP limitations still lead to idle cycles. I
- In coarse grained multithreaded processor, thread switching occurs only when there is a stall and the new thread has a start-up period, there are some fully idle cycles remaining

Drawback:

- It is limited in its ability to overcome throughput losses, especially from shorter stalls.
- This limitation arises from the pipeline start-up costs of coarse-grain multithreading. Because a CPU with coarse-grained multithreading issues instructions from a single thread, when a stall occurs, the pipeline must be emptied or frozen.

Benefit:

- The new thread that begins executing after the stall must fill the pipeline before instructions will be able to complete. Because of this start-up overhead, coarse grained multithreading is much more useful for reducing the penalty of high-cost stalls.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

4.4.3 Simultaneous multithreading (SMT)

- Simultaneous multithreading (SMT) is a variation on multithreading that uses the resources of a multiple-issue, dynamically scheduled processor to exploit TLP at the same time it exploits ILP.
- It has multiple-issue processors and more functional unit parallelism available than a single thread can effectively use.
- Multiple instructions from independent threads can be issued without regard to the dependences among them; the resolution of the dependences can be handled by the dynamic scheduling capability.
- SMT relies on the existing dynamic mechanisms; it does not switch resources every cycle.

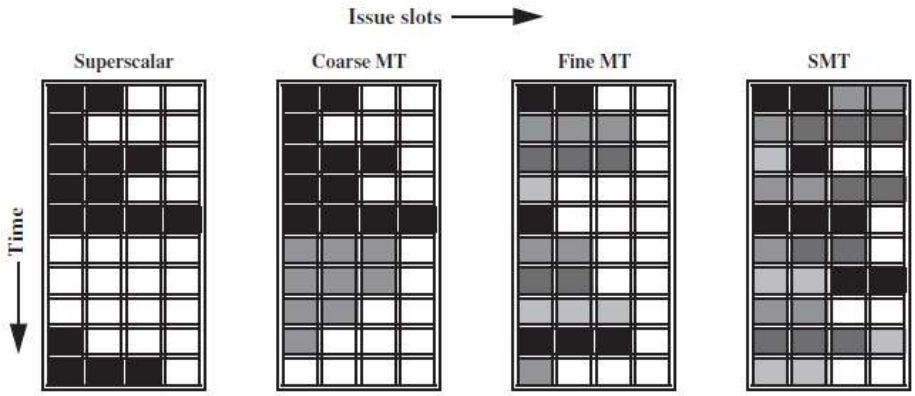

Figure : Four different approaches use the issue slots of a superscalar processor.

- SMT is executing instructions from multiple threads, leaving the hardware to associate instruction slots and renamed registers with their proper threads.
    1. A superscalar with no multithreading support
    2. A superscalar with coarse-grained multithreading
    3. A superscalar with fine-grained multithreading
    4. A superscalar with simultaneous multithreading
- The horizontal dimension represents the instruction issue capability in each clock cycle. The vertical dimension represents a sequence of clock cycles.
- An empty (white) box indicates that the corresponding issue slot is unused in that clock cycle.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- The shades of grey and black correspond to four different threads in the multithreading processors.
- Black is also used to indicate the occupied issue slots in the case of the superscalar without multithreading support.
- In the superscalar without multithreading support, the use of issue slots is limited by a lack of ILP.
- In addition, a major stall, such as an instruction cache miss, can leave the entire processor idle.
- In the SMT case, TLP and ILP are exploited simultaneously, with multiple threads using the issue slots in a single clock cycle. Ideally, the issue slot usage is limited by imbalances in the resource needs and resource availability over multiple threads. In practice other factors can restrict how many slots are used.

Design challenges for an SMT processor:

- Dealing with a larger register file needed to hold multiple contexts
- Not affecting the clock cycle, particularly in critical steps such as instruction issue, where more candidate instructions need to be considered, and in instruction completion, where choosing what instructions to commit may be challenging
- Ensuring that the cache and TLB conflicts generated by the simultaneous execution of multiple threads do not cause significant performance degradation.

4.5 Multi-core Processor

- A multi-core design takes several processor cores and packages them as a single processor.
- The goal is to enable system to run more tasks simultaneously and thereby achieve greater overall system performance.The increasing capacity of a single chip allowed designers to place multiple processors on a single die.
- This approach, initially called on-chip multiprocessing or single-chip multiprocessing, has come to be called multi-core, a name arising from the use of multiple processor cores on a single die.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- In such a design, the multiple cores typically share some resources, such as a second- or third-level cache or memory and I/O buses.
- Recent processors, including the IBM Power5, the Sun T1, and the Intel Pentium D and Xeon-MP, are multi-core and multithreaded.
- Just as using multiple copies of a microprocessor in multiprocessor leverages a design investment through replication, a multi-core achieves the same advantage relying more on replication than the alternative of building a wider superscalar.
- With an MIMD, each processor is executing its own instruction stream. In many cases, each processor executes a different process.
- A process is a segment of code that may be run independently; the state of the process contains all the information necessary to execute that program on a processor.
- In a multi-programmed environment, where the processors may be running independent tasks, each process is typically independent of other processes.
- It is also useful to be able to have multiple processors executing a single program and sharing the code and most of their address space.
- When multiple processes share code and data in this way, they are often called threads
- Today, the term thread is often used in a casual way to refer to multiple loci of execution that may run on different processors, even when they do not share an address space.
- For example, a multithreaded architecture actually allows the simultaneous execution of multiple processes, with potentially separate address spaces, as well as multiple threads that share the same address space.

4.5.1 MIMD Styles

- The existing MIMD multiprocessors fall into two classes depending on the number of processors involved.
    1. Centralized shared memory or Uniform Memory Access time or Shared Memory Processor.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

2. Physically Distributed –Memory Multiprocessor or Decentralized Memory or Memory Module with CPU.

Centralized shared memory:

- A symmetric relationship to all processors
- A uniform memory access (UMA) time for any processor.
- Multi-core processor is composed of two or more independent cores. Manufacture typically integrate the cores into a single integrated circuit die (known as a multi-processor or CMP), or onto multiple dies in a single chip package.

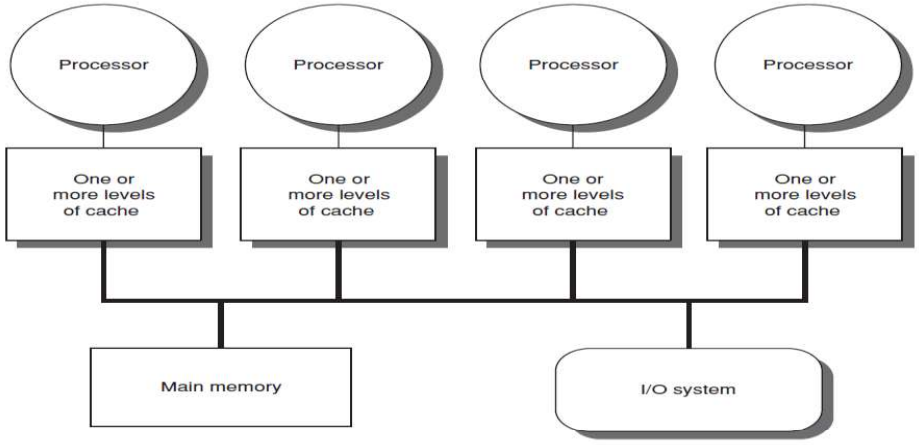

Figure: Basic structure of a centralized shared memory multiprocessor based on a multi-core chip.

- A many core processor is one in which the number of cores is large enough that traditional multiprocessor techniques are no longer efficient-largely due to issues with congestion supplying sufficient instructions and data to the many processors.
- A multiple processor- cache subsystems share the same physical memory, typically with one level of shared cache, and one or more levels of private per-core cache.
- The key architectural property is the uniform access time to all of the memory from all of the processors.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

- In a multichip version the shared cache would be omitted and the bus or interconnection network connecting in the processors to memory would run between chips as opposed to within a single chip.

Advantages:

- Large caches can satisfy the memory demands of a small number of processors.

Disadvantages:

- Scalability problem: Less attractive for large scale processors.
- This type of symmetric shared –memory architecture is currently by far the most popular organization.
- The second group consists of multiprocessors with physically distributed memory. Figure 4.2 shows what these multiprocessors looks like.

Distributed –Memory Multi-core processor:

- The basic architecture of a distributed-memory multiprocessor consists of individual nodes containing a processor, some memory, typically some I/O, and an interface to an interconnection network that connects all the nodes.
- To support larger processor counts, memory must be distributed among the processors rather than centralized; otherwise the memory system would not be able to support the bandwidth demands of a larger number of processors without incurring excessively long access latency.
- With the rapid increase in processor performance and the associated increase in a processor's memory bandwidth requirements, the size of a multiprocessor for which distributed memory is preferred continues to shrink.
- The larger number of processors also raises the need for a High-bandwidth interconnects.
- Both direction networks (i.e., switches) and indirect networks (multidimensional meshes) are used.
- Distributing the memory among the nodes has two major benefits.
    1. First, it is a cost-effective way to scale the memory bandwidth if most of the accesses are to the local memory in the node.

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School

2. Second, it reduces the latency for accesses to the local memory.

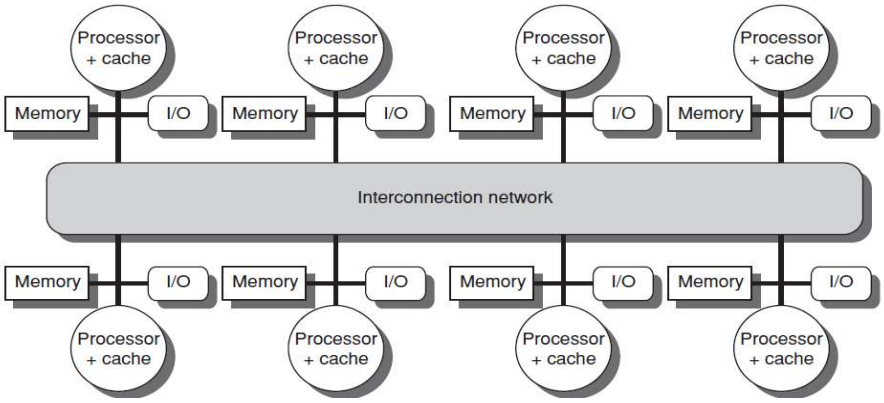

Figure: The basic architecture of a distributed-memory multi-processor .

- The advantages of multi-core processor are
    1. Increased responsiveness and worker productivity.
    2. Improved performance in parallel environments when running computations on multiple processors.
- Multi-core Processor superior to single core processor. On single core processor multithreading can only be used to hide latency.

4.5.2 Classification based on communication

Distributed Shared Memory:

- Distributed Shared Memory (DSM) architecture is a multiprocessor with a shared address space in which communication occurs through a shared address space.

Shared Memory:

- The address space that is shared i.e., the same physical address on two processor refers to the same location in memory.
    1. UMA (Uniform Memory Access Time)
    2. NUMA (Non-Uniform Memory Access Time Multiprocessors).

www.AllAbtEngg.com Study Materials for Anna University, Polytechnic & School