UNIT – IV PARALLELISM

Parallel processing challenges – Flynn‘s classification – SISD, MIMD, SIMD, SPMD, and Vector Architectures - Hardware multithreading – Multi-core processors and other Shared Memory Multiprocessors - Introduction to Graphics Processing Units, Clusters, Warehouse Scale Computers and other Message-Passing Multiprocessors.

1. INSTRUCTION LEVEL PARALLELISM

Instruction level parallelism is the kind of parallelism among instructions. It can exist when instructions in a **sequence** are independent and thus can be executed in parallel by **overlapping**.

PIPELINING is a technique that runs programs faster by ***overlapping the execution of instructions***. This is one example of instruction level parallelism.

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] **Replicate** the internal components of the computer.

1.1. MULTIPLE ISSUE

Multiple issue is a technique used to launch multiple instructions in every pipeline stage. Launching multiple instructions per stage will allow the instruction execution rate to exceed the clock rate or the CPI to be less than 1.

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 decides whether decisions are made at ***compile time*** or during ***execution time***.

1.1.1. STATIC MULTIPLE ISSUE

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

1.1.2. DYNAMIC MULTIPLE ISSUE

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

1.2. Multiple Issue Pipeline

There are two primary and distinct responsibilities that must be dealt with in a multiple issue pipeline such as

[1] Packaging instructions into issue slots
[2] Dealing with data and control hazards

1.2.1. Packaging Instructions Into Issue Slots

Issue slots are the positions from which instructions could issue in a given **clock cycle** and it correspond to positions at the starting blocks for a sprint.

In static issue processors, this process is partially handled by the compiler and in case of **dynamic issue** processors it is dealt with runtime by the processor.

1.2.2. Dealing With Data And Control Hazards

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

1.3. Concept of Speculation

One of the most important methods for finding and exploiting more instruction level parallelism is speculation. It is an approach that allows the compiler or the processor to guess about the properties of an instruction, so to enable execution that begin for other instructions may depend on the speculated instruction.

Speculation is an approach whereby the ***compiler*** or process guesses the outcome of an instruction to remove it as dependence in executing other instructions.

For example, we may ***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 load across a store.

1.3.1. Difficulty with speculation

Difficulty with speculation is it may be wrong. So any speculation mechanism must include 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:

[1] Hardware speculation
[2] Software speculation

In hardware speculation the processor usually buffers the speculative results until it knows they are no longer speculative.

If the speculation is correct, the instructions are completed by allowing the contents of the buffers to be written to the registers or memory:

If the speculation is 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 is incorrect.

Speculation can improve performance when it is done properly and decrease performance when it is done carelessly.

1.4. STATIC MULTIPLE ISSUE PROCESSORS

In static multiple issue, all processors use the compiler to assist with packaging instructions and handling hazards. It will use the issue packet.

Issue packet is set of instructions that issues 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 the mix of instructions that can be initiated in a given clock cycle. So issue packet is a single instruction that allows several operations in certain predefined fields. This approach is called **VERY LONG INSTRUCTION WORD (VLIS).**

Very Long Instruction Word is a style of 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***.

In most static issue processors compiler take some of the responsibility for handling data and control hazards. The responsibilities taken by the compiler are

[1] Static branch prediction
[2] Code scheduling to reduce or prevent all hazards

1.5. DYNAMIC MULTIPLE ISSUE PROCESSORS

Dynamic multiple issue processors are also known as super scalar processors or **simply superscalars**.

Super scalar is an advanced pipelining technique that enables the processor to execute more than one instruction per clock cycle by selecting them during execution.

Achieving good performance on **dynamic multiple issue processors** it requires the compiler to schedule instructions to move dependences and improve the instruction issue rate.

Even with such compiler scheduling, there is an important difference between simple superscalar and a VLIW processor.

In superscalar processor whether the code has scheduled or not it is given to the hardware to **execute correctly**.

In superscalar processor, the compiled code can run always correctly independent of the issue rate or pipeline structure of the processor. In VLIW designs, this has not the case and it requires recompilation when moving across different processor models.

Some **static issue processors** the code can run correctly across different implementations but it requires poorly compilation.

Many **superscalars extend the basic framework of dynamic issue** decisions by including dynamic pipeline scheduling.

Dynamic pipeline scheduling chooses which instructions to execute in a given clock cycle while trying to avoid hazards and stalls.

Consider the simple example to avoid a **data hazard**, consider the following code sequence

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

In that code, even though the sub instruction is ready to execute but it must wait for the `lw` and `addu` instruction to complete first. It takes many clock cycles if memory is slow.

DYNAMIC PIPELINE SCHEDULING allows such hazards to be avoided either fully or partially.

1.5.1. DYNAMIC PIPELINE SCHEDULING

Dynamic pipeline scheduling chooses which instructions to execute next by **REORDERING THE INSTRUCTIONS** it avoid stalls. In such processors, the pipeline is divided into three major units.

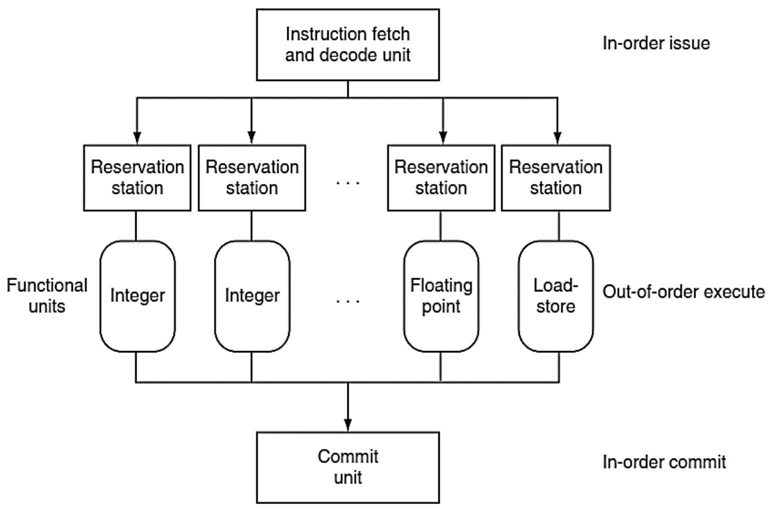
[1] An instruction fetch and issue unit
[2] Multiple functional units
[3] Commit unit

Below figure shows this model. The final step of updating the state is also called retirement or graduation.

The first unit fetches the instructions, decodes them and sends each instruction to a corresponding functional unit for execution.

Each functional unit has a buffer that is called **reservation stations**. It holds the operands and operation.

Once the buffer contains all its operands and the functional units is ready to execute then 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 it into the register file or for a store or into memory.

The buffer in the commit unit called the **reorder buffer**. It holds the results in a dynamically scheduled processor until it is safe to store the results into memory or a register.

Once a register is committed to the register file it can be fetched directly from there just like a **normal pipeline**.

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 register is obtained.

STEP 1

When an instruction issues, it is copied to a reservation station for the appropriate functional unit.

If any operands are available in the **register file** or **reorder buffer** are also **immediately copied** into the reservation station.

The instruction is buffered in the reservation station until all the **operands** and the **functional units** are available.

For the issuing instruction, the register copy of the operand is no longer required and for a write to that register occurred means 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 will produce the result in tracked, when the functional unit eventually produces the result it is copied directly into the waiting reservation station by

passing the registers from functional unit. These two steps use the reorder buffer and the, reservation stations to **implement register renaming**.

1.5.1.1. OUT OF ORDER EXECUTION

The processor executes the instructions in some order that preserves the **DATA FLOW ORDER** of the program. This style of execution is called **OUT OF ORDER EXECUTION**.

1.5.1.2. IN-ORDER COMMIT

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

Dynamic scheduling also use 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 address**, allowing load store reordering and using the commit unit to avoid incorrect speculation.

2. CHALLENGES IN PARALLEL PROCESSING

Parallel processing will increase the performance of processor and it will reduce the utilization time to execute a task. But obtaining the parallel processing is not an easy task. We have difficulty in writing programs to execute a parallel process.

The difficulty with parallelism is ***not in the hardware side*** it is in the software side. Because some 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**.

Developing the parallel processing programs are so harder than the sequential programs because of the following reasons:

First reason is we must get better performance or better energy efficiency from-a parallel processing program on a multiprocessor.

If we would not have efficient energy then we have to use a **sequential program on a uniprocessor** because sequential programming is very simple.

Using uniprocessor we can solve the problem in developing parallel processing programs. Because uniprocessor design techniques such as superscalar and ***out of order execution take advantage of instruction level parallelism***.

Uniprocessor design techniques will reduce the demand for rewriting programs for multiprocessors.

If number of processors level increased means it is more difficult to write parallel processing programs. Because of the following reasons we cannot get parallel processing programs as faster the sequential programs. The reasons are

[1] Scheduling
[2] Partitioning the work into parallel pieces
[3] Balancing the load evenly between the workers
[4] Time to synchronize
[5] Overhead for communication between the parties

These are the five most challenges the developers has to face to write a parallel processing program.

2.1. Scheduling

Scheduling is the method by which threads, processes or data flows are given access to system resources (e.g processor time, communications bandwidth).

Scheduling is done to load balance and share system resources effectively or achieve a **target quality** of service.

Scheduling can be done in various fields among that process scheduling is more important, because in parallel processing, we need to **schedule** the process correctly. Process scheduling can be done in following ways:

[1] Long term scheduling
[2] Medium term scheduling
[3] Short term scheduling
[4] Dispatcher

2.2. Partitioning the work

The task must be ***broken into equal number of pieces*** because otherwise some task may be idle while waiting for the ones with larger pieces to finish. To obtain parallel processing task must be divided into equally to all the processor. Then only we can avoid the idle time of any processor.

2.3. Balancing the load

Load balancing is the process of dividing the amount of work that a computer has to do between two or more processor. So that more work gets done in the same amount of time and in general all process get served faster. ***Work load has to be distributed evenly*** between the processor to obtain parallel processing task.

2.4. Time to synchronize

Synchronization is the most important challenge in parallel processing. Because all the processor have equal work load so it must complete the task within **specific time period**.

For parallel processing program it must have time to synchronization process, because if any process does not complete the task within specific time period then we cannot obtain parallel processing.

2.5. Increasing Scale-Up

Two methods are found to increase the scale up, such methods are

[1] Strong scaling
[2] Weak scaling

2.5.1. Strong scaling

In this method speed up is achieved on a multiprocessor without increasing the size of the problem.

"Strong scaling means measuring speed up while keeping the problem size fixed"

2.5.2. Weak scaling

In this method speed up is achieved on a multiprocessor while increasing the size of the problem proportionally to the increase in the number of processors.

"Weak scaling means the problem size grows proportionally to the increase in the number of processors".

Let's assume that the size of the problem is 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**. By considering the memory hierarchy it is easy to know that weak scaling being easier than strong scaling.

But if the weakly scaled dataset no longer fits in the last level cache of a multicore microprocessor then the resulting performance could be much worse than by using strong scaling.

Depending on the application we can select the scaling approach.

3. Flynn`s Classification

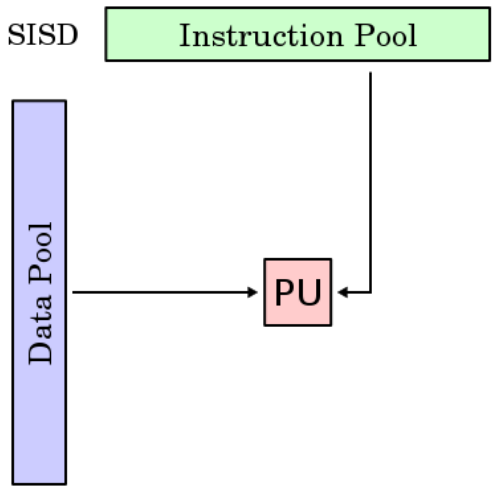
Flynn s classification divides parallel hardware into four broad groups such as

[1] Single Instruction stream Single Data stream (SISD)
[2] Single Instruction stream Multiple Data stream (S1MD)
[3] Multiple Instruction stream Single Data stream (MISD)
[4] Multiple Instruction stream Multiple Data stream (MIMD)

	Single instruction	**Multiple instruction**	**Single program**	**Multiple program**
Single data	SISD	MISD		
Multiple data	SIMD	MIMD	SPMD	MPMD

3.1. Single Instruction, Single Data stream (SISD)

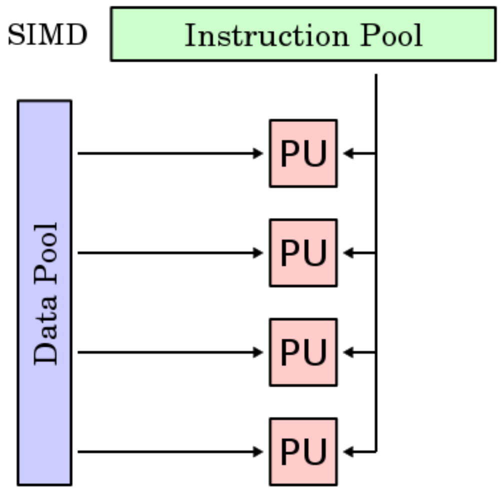
A sequential computer which exploits no parallelism in either the instruction or data streams. Single **control unit** (CU) fetches single **Instruction Stream** (IS) from memory. The CU then generates appropriate control signals to direct single **PROCESSING ELEMENT** (PE) to operate on single **DATA STREAM** (DS) i.e. one operation at a time.



Examples of SISD architecture are the traditional uniprocessor machines like a PC (currently manufactured PCs have multiple cores) or old mainframes.

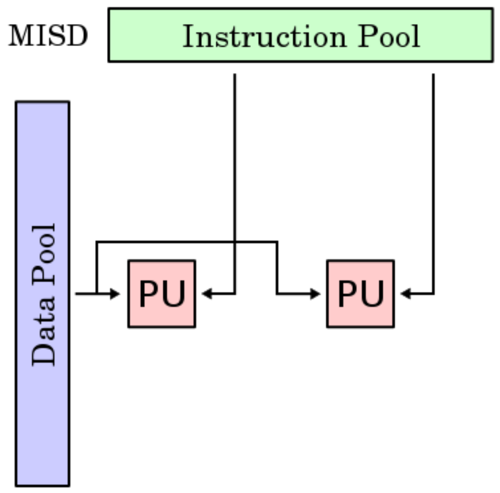
3.2. SINGLE INSTRUCTION, MULTIPLE DATA STREAMS (SIMD)

A computer which exploits multiple data streams against a single instruction stream to perform operations which may be **naturally parallelized**. For example, an array processor or GPU.



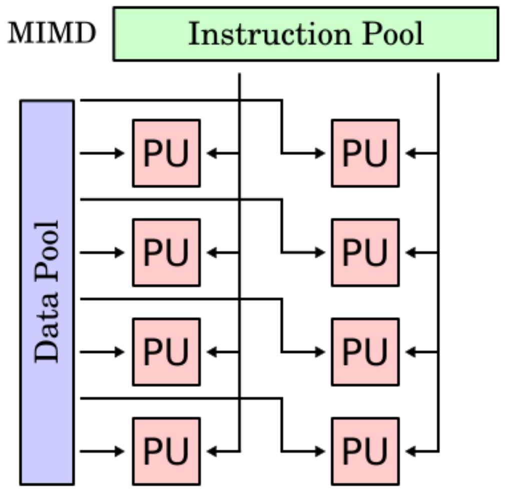
3.3. MULTIPLE INSTRUCTION, SINGLE DATA STREAM (MISD)

Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. **Heterogeneous** systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.



3.4. MULTIPLE INSTRUCTION, MULTIPLE DATA STREAMS (MIMD)

Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single **shared memory space or a distributed memory space**. A multi-coresuperscalar processor is an MIMD processor.



Further MIMD can be divided into two categories

3.4.1. SPMD

Single Program, Multiple Data: multiple autonomous processors simultaneously executing the same program (but at independent points, rather than in the lockstep that SIMD imposes) on different data. Also referred to as ***'Single Process, multiple data'*** - the use of this terminology for SPMD is erroneous and should be avoided, as SPMD is a parallel execution model and assumes multiple cooperating processes executing a program. SPMD is the most common style of parallel programming.

3.4.2. MPMD

Multiple Program, Multiple Data: multiple autonomous processors simultaneously operating at least 2 independent programs. Typically such systems pick one node to be the "**host**" ("the explicit host/node programming model") or "**manager**" (the "Manager/Worker" strategy), which runs one program that farms out data to all the other nodes which all run a second program.

3.5. VECTOR ARCHITECTURE

- SIMD computers operate on vectors of data and it uses vector architectures, the vector architectures pipelined the ALU to get good performance at lower cost.
- The basic philosophy of vector architecture is to collect data elements from memory, put them in order into a large set of registers, operate on them sequentially in registers using pipelined execution units, and then write the results back to memory.
- A key feature of vector architectures is then a set of vector registers. Thus, a vector architecture might have 32 vector registers, each with 64-bit elements.

3.5.1. VECTOR VERSUS SCALAR

- Vector instructions have several important properties compared to conventional instruction set architectures, which are called scalar architectures in this context:
    - A single vector instruction specifies a great deal of work it is equivalent to executing an entire loop. The instruction fetch and decode bandwidth needed is dramatically reduced.
    - By using a vector instruction, the compiler or programmer indicates that the computation of each result in the vector is independent of the computation of other results in the same vector, so hardware does not have to check for data hazards within a vector instruction.
    - Vector architectures and compilers have a reputation of making it much easier than when using MIMD multiprocessors to write efficient applications when they contain data-level parallelism.
    - Hardware need only check for data hazards between two vector instructions once per vector operand, not once for every element within the vectors.
    - Reduced checking can save energy as well as time. Vector instructions that access memory have a known access pattern. If the vectors elements are all adjacent, then fetching the vector from a set of heavily interleaved memory banks works very well. Thus, the cost of the latency to main memory is seen only once for the entire vector, rather than once for each word of the vector.

3.5.2. VECTOR VERSUS MULTIMEDIA EXTENSIONS

- Like multimedia extensions found in the x86 AVX instructions, a vector instruction specifies multiple operations.
- Multimedia extensions typically specify a few operations while vector specifies dozens of operations. Unlike multimedia extensions, the number of elements in a vector operation is not in the opcode but in a separate register.
- Different versions of the vector architecture can be implemented with a different number of elements just by changing the contents of that register and hence retain binary compatibility.
- Vectors support both strided accesses, where the hardware loads every nth data element in memory, and indexed accesses, where hardware finds the addresses of the items to be loaded in a vector register. Indexed accesses are also called gather scatter.
- Like multimedia extensions, vector architectures easily capture the flexibility in data widths, so it is easy to make a vector operation work on 32 64-bit data elements or 64 32-bit data elements or 128 16-bit data elements or 256 8-bit data elements.

vector lane: One or more vector functional units and a portion of the vector register file. Inspired by lanes on highways that increase traffic speed, multiple lanes execute vector operations simultaneously.

- The vector register storage is divided across the lanes with each lane holding every fourth element of each vector register.
- Three vector functional units are FP add,FP multiply and a load store unit.
- Each of the vector arithmetic unit contains four execution pipelines,one per lane which acts in concert to complete a single vector execution.
- Each section of the vector register file only needs to provide enough read and write ports for functional units local to its lane.

VECTOR ARCHITECTURE	MULTIMEDIA EXTENSIONS
It specifies dozens of operations.	It specifies a few operations.
Number of element in a vector operation is not in the opcode.	Number of element in multimedia extension.
Vector has data transfers need to be contiguous.	Multimedia extension has data transfers need to be contiguous.

4. Hardware Multithreading

Hardware multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion try to utilize the hardware resources efficiently. To permit this sharing, the processor must duplicate the independent state of each thread.

For example, each thread would have a separate copy of the register file and the program counter. The memory itself can be shared through the virtual memory mechanisms it 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 processes don't share. Thread switch is more efficient than a process switch. Process includes one or more threads, the address space and the operating system state. Process switch invokes the operating system but not a thread switch.

Hardware multithreading has two main approaches such as

[1] Fine grained multithreading
[2] Coarse grained multithreading

4.1. Fine Grained Multithreading

Fine grained multithreading is a version of hardware multithreading that implies switching between threads after every instruction.

This interleaving is done in a round robin fashion, skipping any threads that are stalled at that clock cycle. To make this multithreading in practical, the processor must be able to switch threads on every clock cycle.

4.1.1. Advantage

It can hide the throughput losses that arise from both short and long stalls because instruction from other threads can be executed when one thread stalls.

4.1.2. Disadvantage

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

4.2. Coarse Grained Multithreading

Coarse grained multithreading is a version of hardware multithreading that implies switching between thread only after significant events such as last level cache miss.

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.

4.2.1. 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 grained multithreading. Because a processor with coarse grained multithreading issues instructions from a single thread.

4.2.2. Benefit

The new thread will begins executing after the stall must fill the pipeline before instructions will be able to complete.

Due to this start-up overhead, coarse grained multithreading is more useful for reducing the penalty of high cost stalls.

4.3. Simultaneous Multithreading (SMT)

Simultaneous multithreading is a variation on hardware multithreading that uses the resources of a multiple issue, dynamically scheduled pipelined processor to exploit thread level parallelism at the same time it exploits instruction level parallelism.

It has multiple processors and more functional unit parallelism available than most single threads can effectively use.

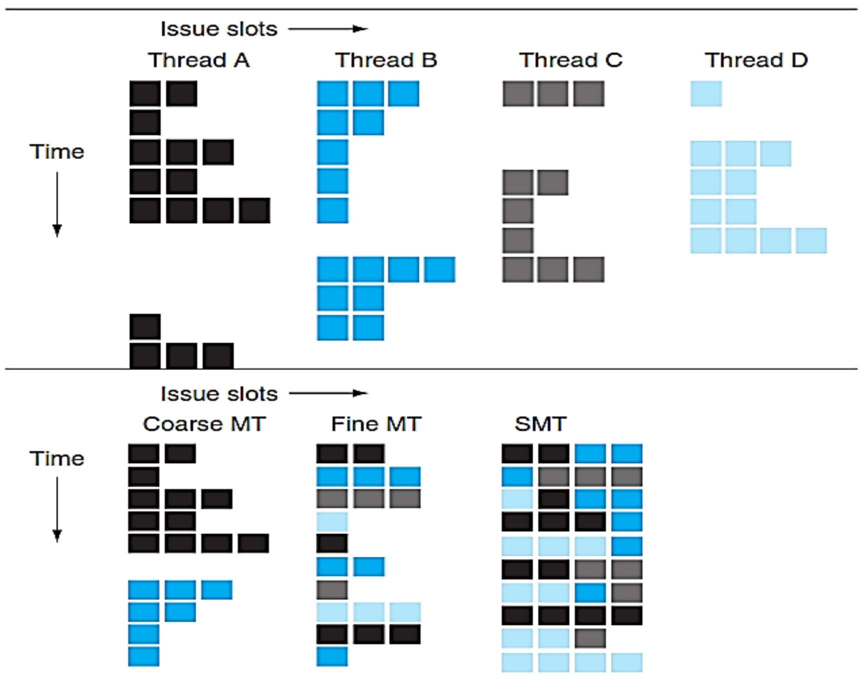
It has register renaming and dynamic scheduling policy with these features the following task can be obtained.

Multiple instructions, from independent threads can be issued without regard to the dependences among them and 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.

SMT is always executing instructions from multiple threads, leaving it up to the hardware to associate instruction slots and renamed registers with their proper threads:

The top portion shows how four threads execute independently on a superscalar with no multithreading support.



The bottom portion shows how the four threads would be combined to execute on the processor more efficiently using three multithreading options.

[1] A superscalar with coarse grained multithreading
[2] A superscalar with fine grained multithreading
[3] A superscalar, with simultaneous multithreading

The horizontal dimension represents the instruction issue capability in each clock cycle and vertical dimension represents a sequence of clock cycles.

In the superscalar without hardware multithreading (top portion) support the use of issue slots is limited by a lack of instruction level parallelism. A major stall such as an instruction cache miss can leave the entire processor.

4.4. COARSE GRAINED MULTITHREADING

In the coarse grained multithreading superscalar, the long stalls are hidden by switching to another thread that uses the resources of the processor. It will reduce the number of completely idle clock cycle and the pipeline start up overhead still leads to idle cycles.

4.5. FINE GRAINED MULTITHREADING

In fine grained multithreading the interleaving of threads mostly eliminates idle clock cycles. Because only a single thread issues instructions in a given clock cycle.

In these multithreading limitations in instruction level parallelism will lead to idle slots within some clock cycles.

4.6. SIMULTANEOUS MULTITHREADING

In the SMT thread level parallelism and instruction level parallelism both are exploited 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, but in practice other factors can restrict how many slots are used.

Multiple threads can utilize the resources of a single processor more effectively.

5. MULTICORE PROCESSORS

Multiple threads can utilize the resources of a single processor more effectively.

Hardware multithreading improved the efficiency of processor but it has big challenge to deliver on the performance potential of **MOORE'S LAW** by efficiently programming the increasing number of processors per chip.

Rewriting old programs to run well on parallel hardware is more difficult. Computer designers must do something to overcome these difficulties.

Solution for above program is using a **single physical address space for all processors** so that programs need not concern themselves with where their data is and programs may be executed in parallel.

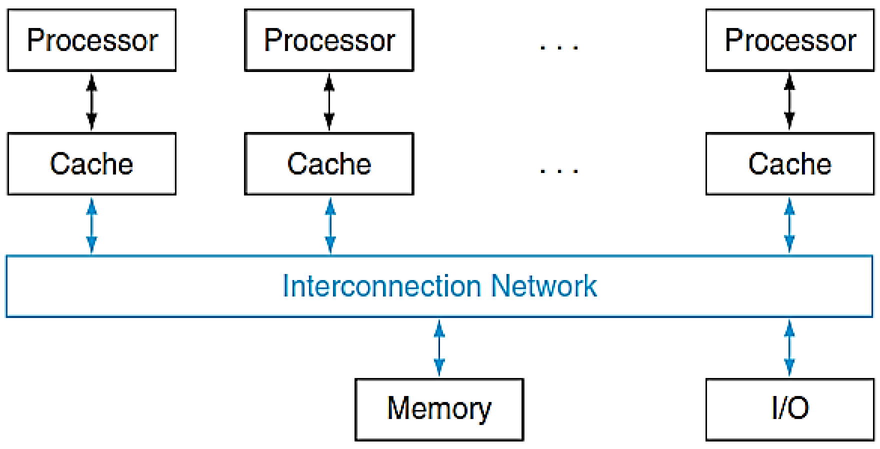
In this approach, all variables of a program can be made **available at any time to any processor**. Using separate address space per processor also we can solve above problem.

When the physical address space is common then the hardware typically provides cache coherence to give a consistent view of the shared memory.

Shared memory multiprocessor allows programmer to use single physical address space across all processors which is always suitable for multicore chips.

In shared address multiprocessor, processors can communicate through variables in memory with all processors capable of **accessing any memory location** via loads and stores.

Below figure shows the classic organization of a shared memory multiprocessor. It can run independent jobs in their own **VIRTUAL ADDRESS SPACES**even if they all share a physical address space.



5.1. Types of Single Address Space Multiprocessor

Single address space multiprocessor comes in two styles such as

[1] Uniform Memory Access (UMA)
[2] Non Uniform Memory Access (NUMA)

Uniform Memory Access is a multiprocessor in which **latency to any word in main memory is same** no matter which processor requests the access.

Non Uniform Memory Access is a type of single address space multiprocessor in which **some memory accesses are much faster than others** depending on which proceed: asks for which word.

Sl. No.	Uniform Memory Access	Non Uniform Memory Access
1	Programming challenges are easy.	Programming challenges are hard.
2	UMA machines can scale small sizes.	NUMA machines can scale to larger sizes.
3	It has higher latency.	It has lower latency to nearby memory.

5.2. Synchronization

As processors operating in parallel will normally share data, they also need to coordinate when operating on shared data otherwise one processor can start working on data before another is finished with it. This coordination is called synchronization.

Synchronization is the process of coordinating the behaviour of two or more; processes which may be running on different processors.

When sharing is supported with a single address space there must be a **separate mechanism for synchronizations** such as called **LOCK**.

Lock is a synchronization device that allows access to data to only one processor at time and other processors interested in shared, data must wait until the original processor unlocks the variable.

6. INTRODUCTION TO GRAPHICAL PROCESSING UNITS

- A major driving force for improving graphics processing was the computer game industry, both on PC's and in dedicated game consoles such as the sony playstation.
- Moreover there are increasing demands for computer generated images for TV advertisements and movies.
- Moore's Law increased the number of transistors available to microprocessors, it therefore made sense to improve graphics processing.As the graphics processors increased in power, they earned the name Graphics Processing Units or GPUs to distinguish themselves from CPUs.
- Here are some of the key characteristics as to how GPUs vary from CPUs:
  - GPUs are accelerators that supplement a CPU, so they do not need be able to perform all the tasks of a CPU.
  - The GPU problems sizes are typically hundreds of megabytes to gigabytes, but not hundreds of gigabytes to terabytes.
  - GPU's are highly multi threaded.
  - GPU's use thread switching to hide memory latency.
  - CPU supports for sequential coding while GPU support for parallel coding.

6.1. AN INTRODUCTION TO THE NVDIA GPU ARCHITECTURE

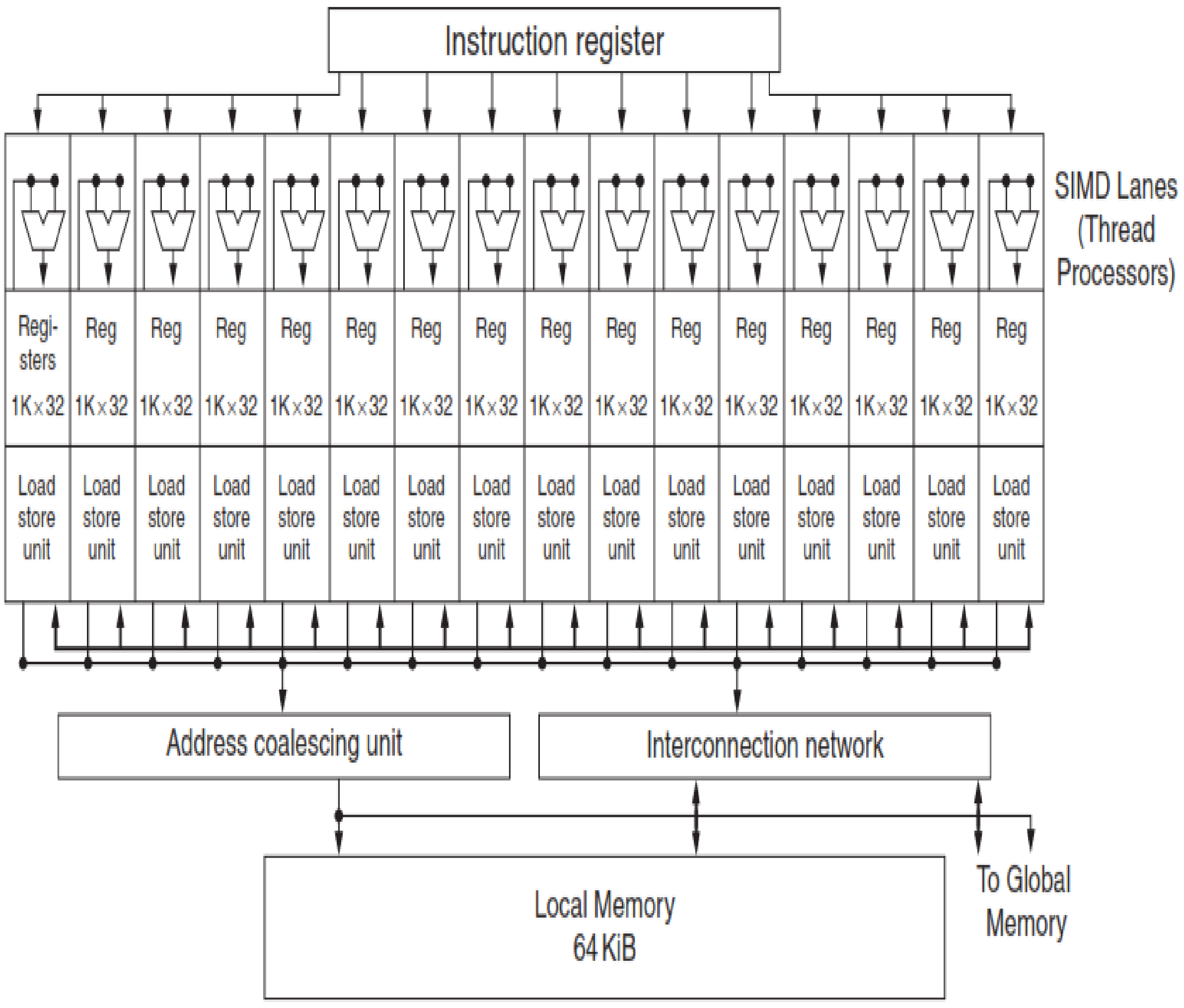
- NVIDIA systems as our example as they are representative of GPU architectures.
- Like vector architectures, GPUs work well only with data-level parallel problems.
- Unlike most vector architectures, GPUs also rely on hardware multithreading within a single multi-threaded SIMD processor to hide memory latency.
- A multithreaded SIMD processor is similar to a Vector Processor, but the former has many parallel functional units instead of just a few that are deeply pipelined, as does the latter.
- For example, NVIDIA has four implementations of the Fermi architecture at different price points with 7, 11, 14, or 15 multithreaded SIMD processors.

SIMD THREAD SCHEDULER:

- The SIMD Thread Scheduler includes a controller that lets it know which threads of SIMD instructions are ready to run, and then it sends them off to a dispatch unit to be run on the multithreaded SIMD processor.
- It is identical to a hardware thread scheduler in a traditional multithreaded processor except that it is scheduling threads of SIMD instructions.
- Thus, GPU hardware has two levels of hardware schedulers:

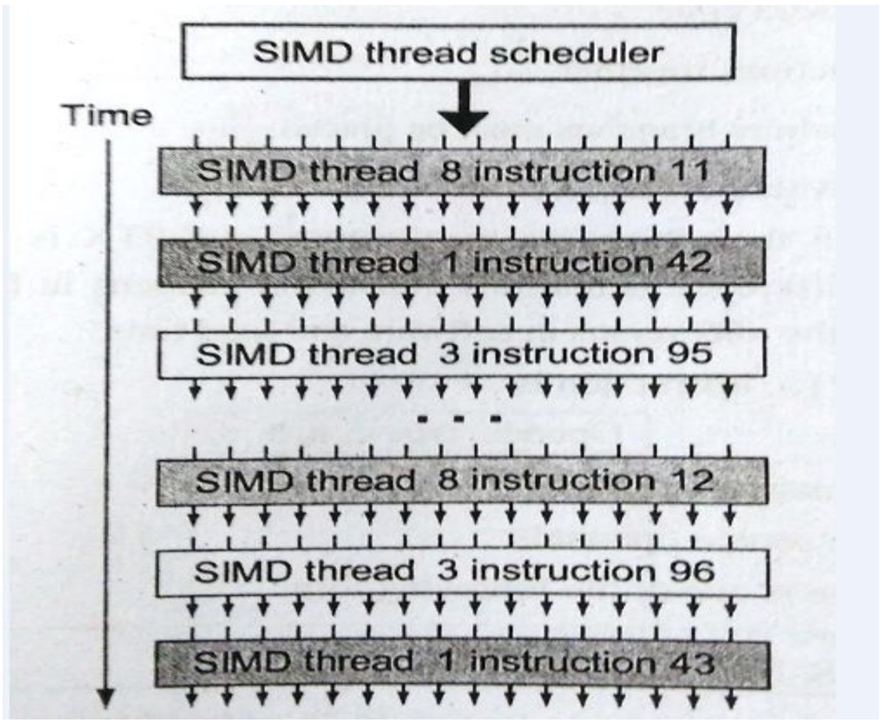
  1. The Thread Block Scheduler that assigns blocks of threads to multithreaded SIMD processors, and

2. The SIMD Thread Scheduler within a SIMD processor, which schedules when SIMD threads should run.



- ➢ The number of lanes per SIMD processor varies across GPU generations.
- ➢ Each thread of SIMD instruction is executed in lock step and scheduled at the beginning.
- ➢ The SIMD thread scheduler can pick whatever thread of SIMD instruction is ready and need not stick with the next SIMD instruction in the sequence within a thread.
- ➢ The scoreboard is needed because memory access instructions can take an unpredictable number of clock cycles.
- ➢ The scheduler selects a ready thread of SIMD instruction and issues an instruction synchronously to all the SIMD lanes executing the SIMD thread.
- ➢ Because of threads of instructions are independent and scheduler may select a different SIMD thread each time.

- Each multithreaded SIMD processor must load 32 elements of two vectors from memory into registers to perform the multiple by reading and writing registers and store the product back from registers into memory.


SIMD thread scheduler
Time
SIMD thread 8 instruction 11
SIMD thread 1 instruction 42
SIMD thread 3 instruction 95
SIMD thread 8 instruction 12
SIMD thread 3 instruction 96
SIMD thread 1 instruction 43

6.2. NVIDIA GPU MEMORY STRUCTURES

- GPU memory is shared by all grids and local memory is shared by all threads of SIMD instructions within a thread block.

Private Memory:

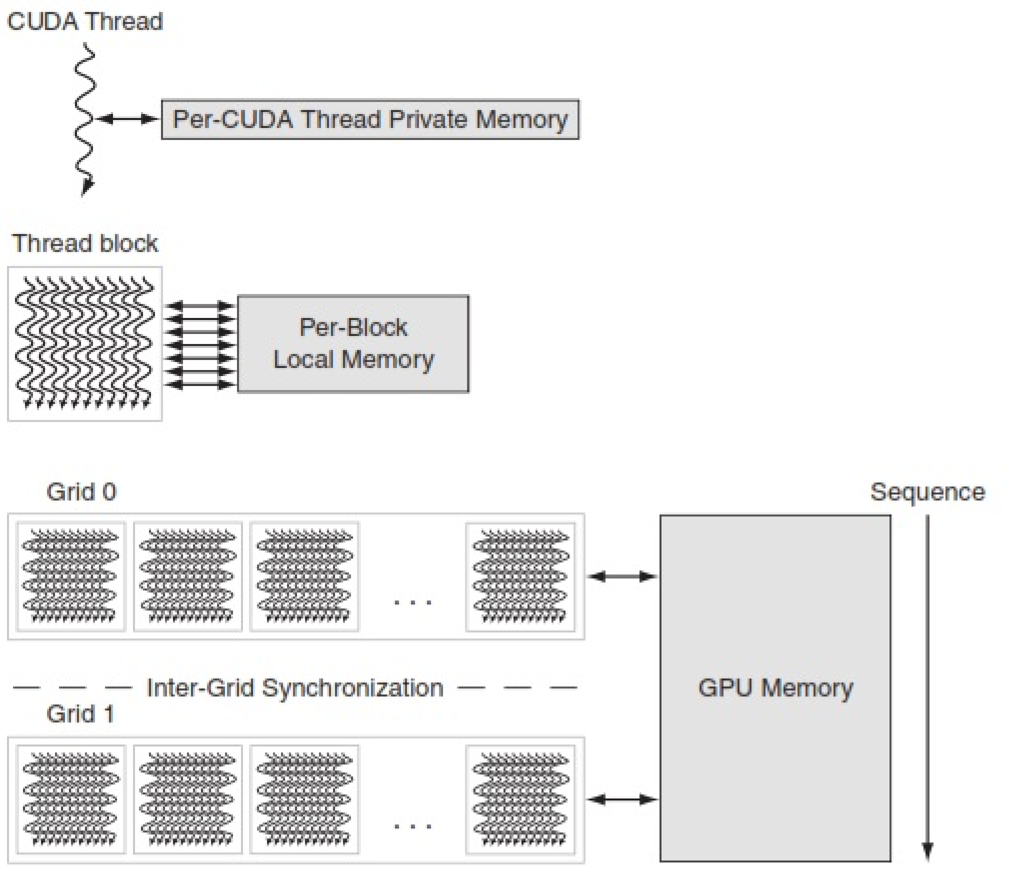
It is used for the stack frame for spilling registers and for private variables that don't fit in the registers. SIMD lanes do not share private memories.

Local Memory:

It is shared by the SIMD lanes within a multithreaded SIMD processor, but this memory is not shared between multithreaded SIMD processors.

GPU Memory:

- It is an Off-chip DRAM shared by the whole GPU and all thread blocks GPU memory.
- The system Procssor, called the host can read or write GPU memory.Local memory is unavailable to the host and it is private to each multithreaded SIMD processor private memories are unavailable to the host.
- To improve memory bandwidth and reduce overhead, data transfer instructions coalesce individual parallel thread requests from the same SIMD thread together into a single memory block requests when the addresses fall into the same block.



6.3. PUTTING GPU'S INTO PERSPECTIVE

- At a high level, multicore computers with SIMD instruction extensions do share similarities with GPUs.
- Both are MIMDs whose processors use multiple SIMD lanes, although GPUs have more processors and many more lanes.
- Both use hardware multithreading to improve processor utilization, although GPUs have hardware support for many more threads.
- Both use caches, although GPUs use smaller streaming caches and multicore computers use large multilevel caches that try to contain whole working sets completely.
- Both use a 64-bit address space, although the physical main memory is much smaller in GPUs. While GPUs support memory protection at the page level, they do not yet support demand paging. SIMD processors are also similar to vector processors.
- The biggest difference is multithreading, which is fundamental to GPUs and missing from most vector processors.
- GPUs and CPUs do not go back in computer architecture genealogy to a common ancestor; there is no Missing Link that explains both.

7. MESSAGE PASIING MULTIPROCESSORS

- Message passing is defined as communication between multiple processors by explicitly sending and receiving information. It has two kinds of routine such as,
  - Send message routine
  - Receive message routine.

  Send message routine: It is used by a processor in machines with private memories to pass a message to another processors.

  Receive message routine: It is used by a processor in machines with private memories to pass a message to another processors.
- The alternative approach to sharing an address space is for the processors to each have their own private physical address space.
- This alternative multiprocessor must communicate via explicit message passing, which traditionally is the name of such style of computers.
- Provided the system has routines to send and receive messages, coordination is built in with **message passing**, since one processor knows when a message is sent, and the receiving processor knows when a message arrives.
- If the sender needs confirmation that the message has arrived, the receiving processor can then send an acknowledgment message back to the sender.

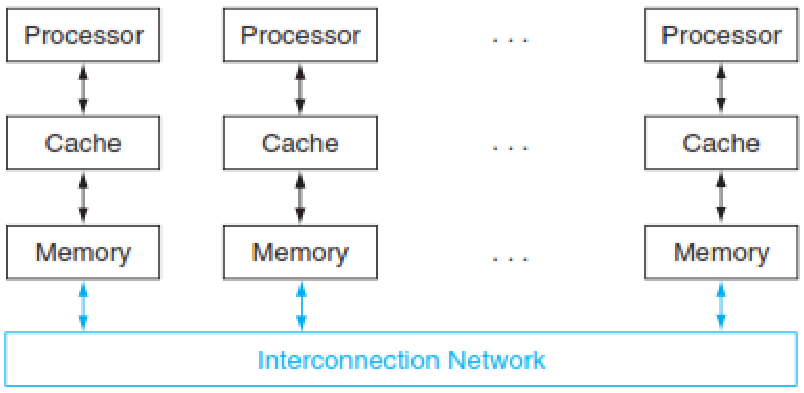


FIGURE 6.13Classic organization of a multiprocessor with multiple private address spaces, traditionally called a message-passing multiprocessor.

- There have been several attempts to build large-scale computers based on high-performance message-passing networks, and they do offer better absolute communication performance than clusters built using local area networks. Indeed, many supercomputers today use custom networks.
- The problem is that they are much more expensive than local area networks like Ethernet.

- Few applications today outside of high performance computing can justify the higher communication performance, given the much higher costs.

8. CLUSTERS

- Clusters Collections of computers connected via I/O over standard network switches to form a message-passing multiprocessor.
- In particular, task-level parallelism and applications with little communication like Web search, mail servers, and file servers do not require shared addressing to run well.
- As a result, clusters have become the most widespread example today of the message-passing parallel computer.
- cluster consists of independent computers connected through a local area network, it is much easier to replace a computer without bringing down the system in a cluster than in an shared memory multiprocessor.
- Fundamentally, the shared address means that it is difficult to isolate a processor and replace it without heroic work by the operating system and in the physical design of the server.
- It is also easy for clusters to scale down gracefully when a server fails, thereby improving dependability.
- Since the cluster software is a layer that runs on top of the local operating systems running on each computer, it is much easier to disconnect and replace a broken computer.
- Given that clusters are constructed from whole computers and independent, scalable networks, this isolation also makes it easier to expand the system without bringing down the application that runs on top of the cluster.
- Their lower cost, higher availability, and rapid, incremental expandability make clusters attractive to service Internet providers, despite their poorer communication performance when compared to large-scale shared memory multiprocessors.

9. WAREHOUSE SCALE COMPUTERS

- Internet services, such as those described above, necessitated the construction of new buildings to house, power, and cool 100,000 servers.
- It is classified as just large clusters, their architecture and operation are more sophisticated.
- They act as one giant computer and cost on the order of $150Mfor the building, the electrical and cooling infrastructure, the servers, and the networking equipment that connects and houses 50,000 to 100,000 servers. We consider them a new class of computer, called **Warehouse-Scale Computers (WSC).**
- While they share some common goals with servers, WSCs have three major distinctions:
  i. ***Ample, easy parallelism:***

- A concern for a server architect is whether the applications in the targeted marketplace have enough parallelism to justify the amount of parallel hardware.
- A WSC architect has no such concern.
- First, batch applications like Map Reduce benefit from the large number of independent data sets that need independent processing, such as billions of Web pages from a Web crawl.
- Second, interactive Internet service applications, also known as Soft Request-Level Parallelism, as many independent efforts can proceed in parallel naturally with little need for communication or synchronization.

ii. *Operational Costs Count:*

- Traditionally, server architects design their systems for peak performance within a cost budget and worry about energy only to make sure they don't exceed the cooling capacity of their enclosure.
- They usually ignored operational costs of a server, assuming that they pale in comparison to purchase costs.

iii. *Scale and the Opportunities/Problems Associated with Scale:*

- To construct a single WSC, you must purchase 100,000 servers along with the supporting infrastructure, which means volume discounts. Hence, WSCs are so massive internally that you get economy of scale even if there are not many WSCs.
- These economies of scale led to cloud computing, as the lower per unit costs of a WSC meant that cloud companies could rent servers at a profitable rate and still be below what it costs outsiders to do it themselves.
- Moore's Law and the increasing number of cores per chip, we now need networks inside a chip as well, so these topologies are important in the small as well as in the large.

PART – A

INSTRUCTION LEVEL PARALLELISM

1. **What is meant by multiprocessor?**

   Multiprocessor is a computer system with at least two processors. This computer is contrast to a uniprocessor.

2. **What is ILP?[Nov/Dec-2015][Nov/Dec-2016][april/May-2017].**

   Instruction level parallelism is the kind of parallelism among instructions. It can exist when instructions in a **sequence** are independent and thus can be executed in parallel by **overlapping**.

3. **State the need for instruction level parallelism.[May/June-2016]**

   Instruction level parallelism can be used to improve the program execution performance by causing individual machines operations to execute in parallel.

4. **What is meant by task level parallelism?**

   Task level parallelism also called as process level parallelism. Task level parallelism utilizing multiple processors by running independent programs simultaneously.

5. **What is parallel processing program?**

   Parallel processing program is a single program that runs on multiple processors simultaneously.

6. **What is cluster?**

   Cluster is a set of computers connected over a local area network that function as a single large multiprocessor;

7. **What is a multicore microprocessor?**

   A microprocessor containing multiple processors (cores) in a single integrated circuit. Virtually all microprocessors today in desktops and servers are multicore.

8. **What is shared memory multiprocessor?**

   Shared memory multiprocessor is a parallel processor with a single physical address space.

9. **What are the challenges includes in parallel programming?**

   Parallel programming challenges includes scheduling, partitioning the work into parallel pieces, balancing the loud evenly between the workers, time to synchronize and overhead for communication between the parties.

10. How to get good speed up on a multiprocessor?

To achieve a good speed up on a multiprocessor problem size must be fixed and the problem size is increased means it is hard to get good speed up.

PARALLEL PROCESSING CHALLENGES

11. Write two methods used to increase the scale up.

Two methods to increase the scale up are

[1] Strong scaling

[2] Weak scaling

12. What is strong scaling?

In these methods speed up achieved on a multiprocessor without increasing the size of the problem.

"Strong scaling means measuring speed up while keeping the problem sizefixed".

13. What is weak scaling?

In this method speed up is achieved on a multiprocessor while increasing the size of the problem proportionally to the increase in the number of processors.

14. Differentiate between strong scaling and weak scaling [May/June 2015][Nov/Dec-2017].

Strong Scaling	**Weak Scaling**
To run a program faster	To run a bigger program
Speed up achieved on a multiprocessor without increasing the size of the problem	Speed up is achieved on a multiprocessor while increasing the size of the problem

FLYNN`S CLASSIFICATION

15. Write Flynn's classification for parallel hardware. [Nov/Dec 2014]

Flynn's classification divides parallel hardware into four groups based on the number of instruction streams and the number of data streams.

[1] Single Instruction stream Single Data stream (SISD)

[2] Single Instruction stream Multiple Data stream (SIMD)

[3] Multiple Instruction stream Single Data stream (MISD)

[4] Multiple Instruction stream Multiple Data stream (MIMD)

16. What is SISD?

Single Instruction stream Single Data stream is a uniprocessor in $m_l = m_d = 1$. Conventional machines with a single CPU capable only of scalar arithmetic fall into this category.

17. What is SIMD?

Single Instruction stream Multiple Data streams the same instruction is applied to many data stream as in a vector processor. Here m1 = 1, m_D> 1, it has single program control unit and many independent execution units.

18. What are the advantages of SIMD?

- Cost of the control unit over dozens of execution unit.
- It has reduced instruction bandwidth and space.
- It needs only one copy of the code that is being executed simultaneously.

19. What are drawbacks of SIMD?

SIMD method is not suitable for case or switch condition data because depending on what data it has execution unit must perform a different operation.

20. What is MISD?

Multiple Instruction stream Single Data stream processor is a stream processor that perform a series of computations on a single data stream in a pipelined fashion. Here m1 > 1, m_D = 1, fault tolerant computers where several CPU's process the same data using different programs are MISD.

21. What is MIMD?

Multiple Instruction stream Multiple Data stream is a multiprocessors, which are computers with more than one CPU and the ability to execute several programs simultaneously.

HARDWARE MULTITHREADING

22. What is data level parallelism?

Data level parallelism is a kind of parallelism achieved by performing the sameoperation on independent data.

23. What is basic principle of vector architecture?

Basic principle of vector architecture is to collect data elements from memory, put the data into a large set of registers, operate on them sequentially in registers using pipelined execution units and then write the results back to memory.

24. What is strip mining?

In vector architecture if the loops are larger then we add book keeping code to iterate full length vector operations and to handle the leftovers. This process is called strip mining.

25. What is vector lane?

Vector lane is one or more vector functional units and a portion of the vector register file. Inspired by lanes on highways that increase traffic speed and multiple lanes execute vector operations simultaneously.

26. What is hardware multithreading? [Nov/Dec 2014]

Hardware multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion to try to utilize the hardware resources efficiently.

27. What are the approaches involved in hardware multithreading process?

There are two main approaches to hardware multithreading such as

[1] Fine grained multithreading

[2] Coarse grained multithreading

28. What is meant by thread?

Thread is a lightweight process which includes the program counter, the register state and stack. It shares a single address space.

29. Define process.

Process is a task on currently being execution. It includes one or more threads, the address space and the operating system state. Process switch can invoke the operating system but thread switch cannot do it.

30. What is fine grained multithreading?[May/June-2016][Nov/Dec-2017]

Fine grained multithreading is a version of hardware multithreading that implies switching between threads after every instruction.

31. What is coarse grained multithreading?[Nov/Dec-2017]

Coarse grained multithreading is a version of hardware multithreading that implies switching between thread only after significant events such as last level cache miss.

32. Write the advantages and disadvantages of fine grained multithreading.

ADVANTAGE

It can hide the throughput losses that arise from both short and long stalls because instruction from other threads can be executed when one thread stalls.

DISADVANTAGE

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

33. Write the advantages and disadvantages of coarse grained multithreading.

ADVANTAGE

It is more useful for reducing the penalty of high cost stalls.

DISADVANTAGE

It is limited in its ability to overcome throughput losses, especially from shorter stalls.

34. What is simultaneous multithreading?

Simultaneous multithreading is a variation on hardware multithreading that uses the resources of-a multiple issue, dynamically scheduled micro architecture.

35. Distinguish implicit multithreading and explicit multithreading.

Implicit multithreading:

- Implicit multithreading is concurrent execution of multiple threads extracted from single sequential program.
- Implicit threads defined statically by compiler or dynamically by hardware.

Explicit Multithreading:

- It is a computer science paradigm for building and programming parallel computers around the parallel random access machine (PRAM) parallel computational model.

36. How many types in single address space multiprocessor?

Single address space multiprocessor comes in two styles such as

[1] Uniform Memory Access (UMA)

[2] Non Uniform Memory Access (NUMA)

37. Define uniform memory access (UMA).

Uniform Memory Access is a multiprocessor in which latency to any word in main memory is same no matter which processor requests the access.

38. What is non uniform memory access (NUMA)?

Non Uniform Memory Access is a type of single address space multiprocessor in which some memory accesses are much faster than others depending on which processor asks for which word.

39. What are the differences between UMA and NUMA? [May/June 2015]

S. No	Uniform Memory Access	-Non Uniform Memory Access
1	Programming challenges are easy.	Programming challenges are hard.
2	UMA machines can scale small sizes.	NUMA machines can scale to larger sizes.
3.	It has higher latency.	It has lower latency to nearby memory.

MULTICORE PROCESSORS

40. Define synchronization.

Synchronization is the process of coordinating the behavior of two or more processes which may be running on different processors.

41. What is meant by lock?

Lock is a synchronization device that allows access to data to only one processor at a time and other processors interested in shared data must wait until the original processor unlocks the variable.

42. What is multiple issue?

Multiple issue is scheme used to place multiple instructions in one clock cycle. It has two types

[1] Static multiple issue

[2] Dynamic multiple issue

43. What is static multiple issue?

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

44. What is dynamic multiple issue?

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

45. What is speculation?

Speculation is an approach in that compiler or process guesses the outcome of an instruction to remove it as a dependence in executing other instructions.

46. What is issue packet?

Issue packet is set of instructions that issues together in one clock cycle and the packet may be determine statically by the compiler or dynamically by the processor.

47. What is VLIW?

VLIW is Very Long Instruction Word, it is a style of instruction set architecture that launches many operations.

All instructions are independent in a single wide instruction with many separateopcode fields.

48. What is loop unrolling?

Loop unrolling is an important compiler technique to get more performance from loops. In unrolling multiple copies of the loop body are made.

49. What is register renaming?

Register renaming is the process of renaming of registers by the compiler or hardware to remove anti-dependences.

50. What is superscalar processor?[nov/Dec-2015]

Superscalar processor is a dynamic multiple issue processor. It is an advanced technique that enables the processor to execute more than one instruction per clock cycle by selecting them during execution.

INTRODUCTION TO GRAPHICS PROCESSING UNITS

51. What is a thread of SIMD instructions?

A traditional thread which contains just SIMD instructions that are executed on a multithreaded SIMD processor is called thread of SIMD instructions.

52. What is the function of SIMD block scheduler?

The Thread Block Scheduler that assigns blocks (blocks of vectorized loop)of threads to multithreaded SIMD processors.

53. What is the function of SIMD thread scheduler?

SIMD thread scheduler schedules and issues threads of SIMD instructions when they are ready to execute, includes a scoreboard to track SIMD thread execution.

CLUSTERS

54. What is clusters?

Clusters Collections of computers connected via I/O over standard network switches to form a message-passing multiprocessor.

55. What is send message routine?

Send message routine: It is used by a processor in machines with private memories to pass a message to another processors.

56. What is receive message routine?

Receive message routine: It is used by a processor in machines with private memories to pass a message to another processors.

PART – B

INSTRUCTION LEVEL PARALLELISM

1. **Explain in detail about Instruction Level Parallelism [Nov/Dec 2014 – 16M]** [Page No:4.1]

PARALLEL PROCESSING CHALLENGES

2. **Explain the challenges in parallel processing method.[May/June-2017][April/May-2018]** [Page No:4.6]

FLYNN`S CLASSIFICATION

3. **Explain the Flynn`s Classification for processors in detail with example.[Nov/Dec-2015][May/june-2016][May/June-2017][Nov/Dec-2017]** [Page No:4.8]
   [OR]
   Discuss about SISD, MIMD, SIMD, SPMD and Vector Systems [May/June 2015 – 16M] Page No:4.8]

HARDWARE MULTITHREADING

1. **Explain in detail the concept of Hardware Multithreading and its types [Nov/Dec 2014 – 8M][nov/Dec-2015][May/June-2016][Nov/Dec-2016]** [Page No:4.13]
   [OR]
   What is Hardware Multithreading? Compare and Contrast Fine Grained Multithreading and Coarse Grained Multithreading [May/June 2015 – 16M] [Page No:4.13]
2. **Explain the four principal approaches to multithreading with necessary diagrams.[May/June-2017]** [Page No:4.13]
3. **Describe simultaneous multithreading(SMT) with an example.[Nov/Dec-2017]** [Page No:4.14]
4. **Compare and contrast fine grained multithreading, coarse grained multi threading and simultaneous multithreading.[May/June-2018]** [Page No:4.13]

MULTICORE PROCESSORS

5. **Explain in detail about multicore processors and how they are different from multiprocessors. [Nov/Dec 2014 – 8M][May/June-2016]** [Page No:4.16]
6. **Discuss shared memory multiprocessors with a neat diagram.[Nov/Dec-2016][May/June-2018]** [Page No:4.16]

GPU

7. **Write a note on GPU architecture.** [Page No:4.18]
8. **Write a note on NVIDIA GPU memory structures.** [Page No:4.20]
9. **Write a note on message-passing multiprocessors.** [Page No:4.22]
10. **Write a note on clusters and warehouse scale computers.[** Page No:4.23]