Instruction Level Parallelism (ILP)

What's ILP

- Architectural technique that allows the overlap of individual machine operations. (add, mul, load, store …)
- Multiple operations will execute in parallel. (simultaneously)
- Goal: Speed Up the execution
- Example:

```
load R1 ← R2              add   R3 ← R3, "1"
add  R3 ← R3, "1"         add   R4 ← R3, R2
add  R4 ← R4, R2          store [R4] ← R0
```

Example: Sequential vs ILP

- Sequential execution (Without ILP)

```
Add r1, r2, r8    4 cycles
Add r3, r4, r7    4 cycles            8 cycles
```

- ILP execution (overlap execution)

```
Add r1, r2, r8
Add r3, r4, r7
  Total of 5 cycles
```

ILP vs Parallel Processing

ILP

- Overlap individual machine operations - Executes in parallel.
- Transparent to the user
- Goal: speed up execution

Parallel Processing

- Separate processors getting separate chunks of the program.
- Nontransparent to the user
- Goal: speed up and quality up

ILP Challenges

1. H/W terminology Data Hazards (RAW, WAR, WAW)
2. S/W terminology Data Dependencies

Data Dependence and Hazards

- Instr_J is data dependent on Instr_I:
    1. Instr_J reads operand written by Instr_I

Types of Dependencies

- Name dependencies
    - Output dependence
    - Anti-dependence
- Data True dependence
- Control Dependence

Name dependences

- **Output dependence**

  Instruction i and j writes the same register or memory location.
  The ordering is preserved to leave the correct value in the register.

  ```
  For Example:  i: add r7,r4,r3
                j: div r7,r2,r8
  ```

- **Anti-dependence**

  Instruction j writes a register or memory location that instruction i reads.

  ```
  For Example : i : add r6,r5,r4
               j: sub r5,r8,r11
  ```

Data Dependences

An instruction *j* is data dependent on instruction *i* if either of the following hold:

- The result of Instruction I may be used by instruction j.
- Instruction j is data dependent on instruction k, and instruction k is data dependent on instruction i.

```
LOOP  LD    F0, 0(R1) -> i

      ADD   F4, F0, F2 -> j

      SD    F4, 0(R1) -> k

      SUB   R1, R1, -8

      BNE   R1, R2, LOOP
```

Control Dependences

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order.

```
if p1 {
     S1;
};
if p2 {
     S2;
}
```

- S1 is control dependent on p1, and S2 is control dependent on p2 but not on p1.

Compiler techniques to increase ILP

- Register Renaming
- Pipeline Scheduling
- Loop Unrolling

Register Renaming

- Instructions involved in a name dependence can execute simultaneously if name used in instructions is changed so instructions do not conflict.

  ▪ Register renaming resolves name dependence for registers either by compiler or by HW.

Code without register renaming

- I: sub r1,r4,r3
- J: add r1,r2,r3
- K: mul r6,r1,r7

Code with register renaming

- I: sub r1,r4,r3
- J: add r8,r2,r3
- K: mul r6,r8,r7

Pipeline Scheduling

- Rearranging and modifying instruction to maximize instruction execution overlap.
- Dynamic pipeline scheduling chooses which instructions to execute next, possibly reordering them to avoid stalls.

Dynamic pipeline scheduling

In such processors, the pipeline is divided into three major units: an instruction fetch and issue unit, multiple functional units and a commit unit. 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, which hold the operands and the operation. As soon as 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, which buffers the result until it is safe to put the result into the register file or, for a store, into memory. The buffer in the commit unit, often called the reorder buffer, is also used to supply operands, in much the same way as forwarding logic does in a statically scheduled pipeline. Once a result is commited to the register file, it can be fetched directly from there, just as in a normal pipeline.

Loop Unrolling

- The loop body is replicated multiple times.
- More instructions in one loop to perform more overlapping.
  - Replicate body

- Remove loop overhead
- Rename registers
- Schedule

Parallel Processing

- Parallel Processing is an efficient form of information processing which emphasizes the exploitation of concurrent events.
  - Concurrency implies parallelism, simultaneity and pipelining.
  - Multiple jobs or programs –multiprogramming
  - Time sharing
  - Multiprocessing
- Parallel Processing requires knowledge of:
  - Algorithms
  - Languages
  - Software
  - Hardware
  - Performance Evaluation
  - Computing Alternatives
- Parallel Processing in UniProcessor:
  - Multiple Functional Units
  - Pipelining, Parallel Adders, Carry Look Ahead Adders
  - Overlapped CPU and I/O operation
  - Use of Hierarchical Memory
  - Balancing Subsystem Bandwidth
  - Multiprogramming and Time Sharing

Parallel Processing - Examples

- Embedded Systems
- Scientific Applications

Flynn's Classification of Computer Architectures (Derived from Michael Flynn, 1972)

- Michael Flynn proposed a classification for computer architectures based on the number of instruction steams and data streams (Flynn's Taxonomy).
- Flynn uses the stream concept for describing a machine's structure
- A stream simply means a sequence of items (data or instructions).
- The classification of computer architectures based on the number of instruction steams and data streams (Flynn's Taxonomy).

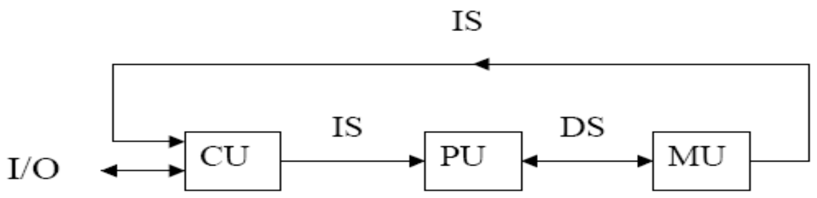
Flynn's Taxonomy

- SISD: Single instruction single data
  - – Classical von Neumann architecture
- SIMD: Single instruction multiple data

- MISD: Multiple instructions single data
  - – Non existent, just listed for completeness
- MIMD: Multiple instructions multiple data
  - – Most common and general parallel machine

SISD (Singe-Instruction stream, Singe-Data stream)

- SISD corresponds to the traditional mono-processor (von Neumann computer). A single data stream is being processed by one instruction stream   OR
- A single-processor computer (uni-processor) in which a single stream of instructions is generated from the program.

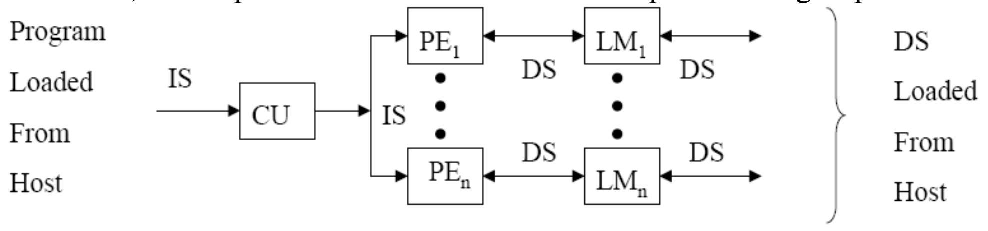


(a) SISD Uniprocessor Architecture

- Captions:
  - ▪ CU -Control Unit; PU –Processing Unit
  - ▪ MU –Memory Unit; IS –Instruction Stream
  - ▪ DS –Date Stream

SIMD (Single-Instruction stream, Multiple-Data streams)

- o Each instruction is executed on a different set of data by different processors i.e multiple processing units of the same type process on multiple-data streams.
- o This group is dedicated to array processing machines.
- o Sometimes, vector processors can also be seen as a part of this group.



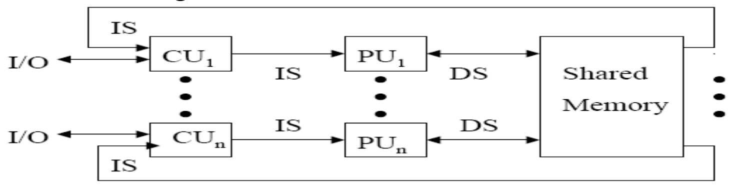
- **(b) SIMD Architecture (with Distributed Memory)**

Captions:

CU - Control Unit	;	**PU - Processing Unit**
MU - Memory Unit	;	**IS - Instruction Stream**
DS - Date Stream	;	**PE – Processing Element**
LM – Local Memory		

MISD (Multiple-Instruction streams, Singe-Data stream)

- Each processor executes a different sequence of instructions.
- In case of MISD computers, multiple processing units operate on one single-data stream .
- In practice, this kind of organization has never been used



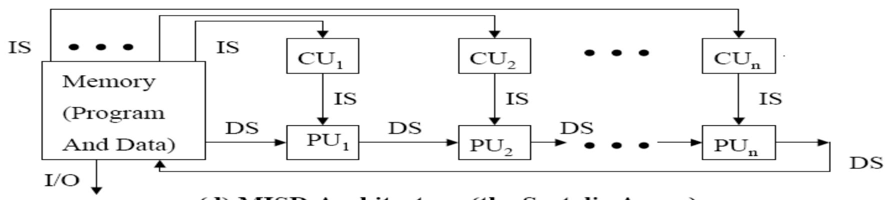
(c) MIMD Architecture (with Shared Memory)

Captions:

CU - Control Unit	;	PU - Processing Unit
MU - Memory Unit	;	IS - Instruction Stream
DS - Date Stream	;	PE – Processing Element
LM – Local Memory		

MIMD (Multiple-Instruction streams, Multiple-Data streams)

- Each processor has a separate program.
- An instruction stream is generated from each program.
- Each instruction operates on different data.
- This last machine type builds the group for the traditional multi-processors. Several processing units operate on multiple-data streams.



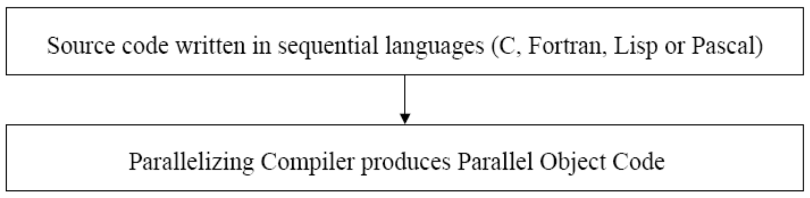
(d) MISD Architecture (the Systolic Array)

Captions:

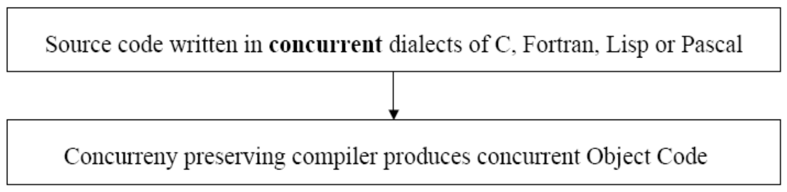
CU - Control Unit	;	PU - Processing Unit
MU - Memory Unit	;	IS - Instruction Stream
DS - Date Stream	;	PE – Processing Element
LM – Local Memory		

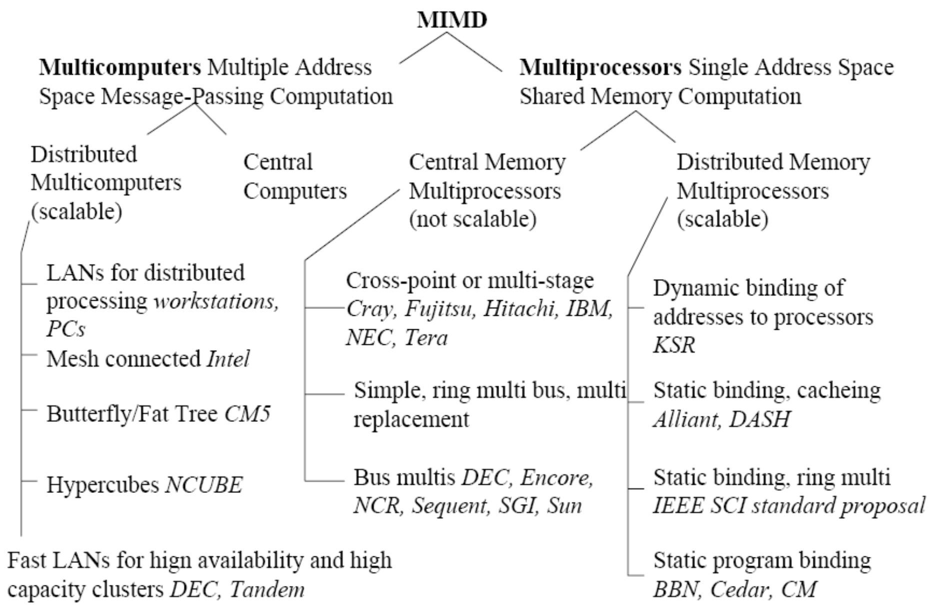
Two Approaches to Parallel Programming

- Implicit Parallelism


Source code written in sequential languages (C, Fortran, Lisp or Pascal)
Parallelizing Compiler produces Parallel Object Code

- Explicit Parallelism


Source code written in **concurrent** dialects of C, Fortran, Lisp or Pascal
Concurreny preserving compiler produces concurrent Object Code


MIMD
Multicomputers Multiple Address Space Message-Passing Computation
Multiprocessors Single Address Space Shared Memory Computation
Distributed Multicomputers (scalable)
Central Computers
Central Memory Multiprocessors (not scalable)
Distributed Memory Multiprocessors (scalable)
LANs for distributed processing *workstations, PCs*
Mesh connected *Intel*
Butterfly/Fat Tree *CM5*
Hypercubes *NCUBE*
Fast LANs for hign availability and high capacity clusters *DEC, Tandem*
Cross-point or multi-stage *Cray, Fujitsu, Hitachi, IBM, NEC, Tera*
Simple, ring multi bus, multi replacement
Bus multis *DEC, Encore, NCR, Sequent, SGI, Sun*
Dynamic binding of addresses to processors *KSR*
Static binding, cacheing *Alliant, DASH*
Static binding, ring multi *IEEE SCI standard proposal*
Static program binding *BBN, Cedar, CM*

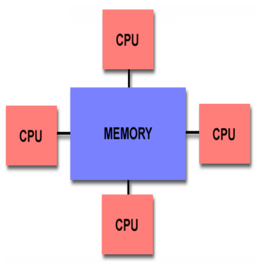
Challenges in Parallel Processing

- Dealing with memory
  - Global Shared Memory
  - Distributed Shared Memory
  - Global shared memory with separate cache for processors

- Potential Hazards:
  - Individual CPU caches or memories can become out of synch with each other. "Cache Coherence"

- Solutions:
  - UMA/NUMA machines
  - Snoopy cache controllers
  - Write-through protocols

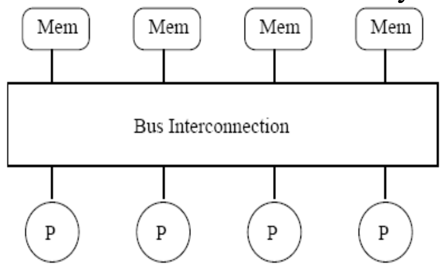
Shared memory architectures



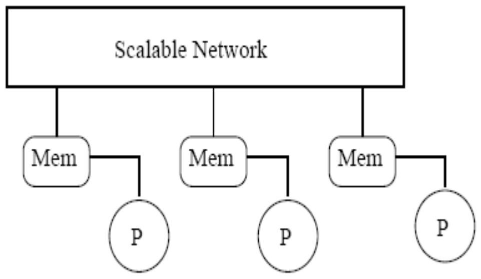
- Multiple CPU's (or cores)
- One memory with a global address space

- May have many modules
- All CPUs access all memory through the global address space
- All CPUs can make changes to the shared memory

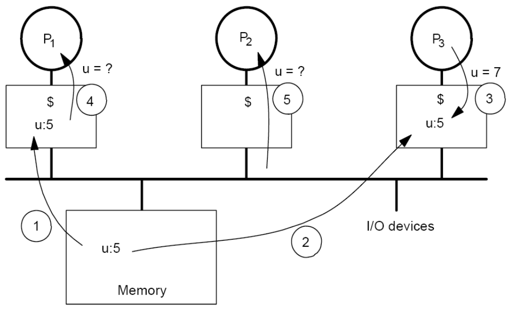
  - One large memory
    - One the same side of the interconnect
      - Mostly Bus
    - Memory reference has the same latency
    - Uniform memory access (UMA)



- Many small memories
- Local and remote memory
- Memory latency is different
- Non-uniform memory access (NUMA)



Cache coherence problem

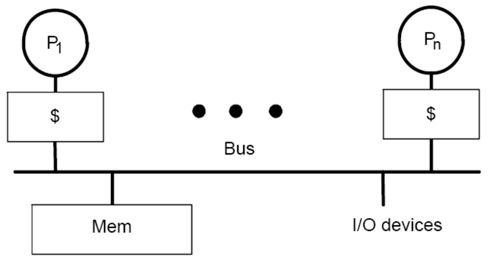


- Due to the cache copies of the memory, different processors will see different values of the same memory location.
- Processors see different values for u after event 3.

- With write-back cache, memory stores the stale date.

Bus Snoopy Cache Coherence protocols

- Memory: Centralized with uniform access time and bus interconnect.
- Example: All Intel Multi Processor machines.



(b) Bus-based shared memory

Bus Snooping idea

- All requests for data are sent to all processors (through the bus).
- Processors snoop to see if they have a copy and respond accordingly.
  - Cache listens to both CPU and BUS.
  - The state of a cache line may change by (1) CPU memory operation, and (2) bus transaction (remote CPU's memory operation).
- Requires broadcast since caching information is at processors.
  - Bus is a natural broadcast medium.
  - Bus (centralized medium) also serializes requests.
- Dominates small scale machines.

Types of snoopy bus protocols

- Write invalidate protocols
  - Write to shared data: an invalidate is sent to the bus (all caches snoop and invalidate copies).

- Write broadcast protocols (typically write through)
  - Write to shared data: broadcast on bus, processors snoop and update any copies.