BIT 325 PARALLEL PROCESSING

- ASSESSMENT
  - CA – 40%
    - TESTS – 30%
    - PRESENTATIONS – 10%
  - EXAM – 60%
- CLASS TIME TABLE
  - SUNDAYS – 16:00 – 18:00
- SYLLUBUS & RECOMMENDED BOOKS
  - CLASS REPRESENTATIVE
  - Lecture Notes: www.Lechaamwe.weebly.com

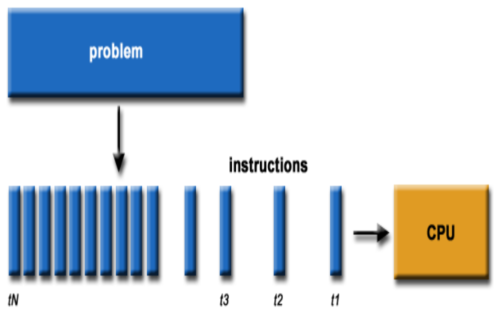
Parallel processing

- Overview
- Clarification of parallel machines
- Some General Parallel Terminology
- Shared memory and message passing

What is Parallel Computing?

- Traditionally, software has been written for ***serial*** computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any moment in time.

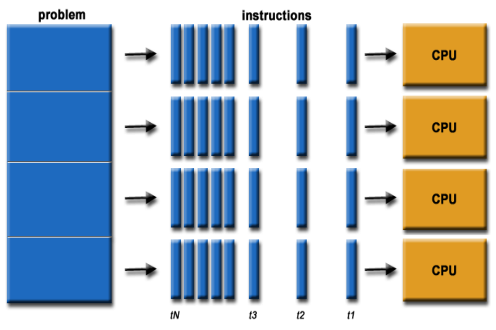
What is Parallel Computing?



What is Parallel Computing?

- ***Parallel computing*** is the simultaneous use of multiple computer resources to solve a computational problem:
  - To be run using multiple CPUs
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute simultaneously on different CPUs

What is Parallel Computing?



What is Parallel Computing?

- The compute resources might be:
  - A single computer with multiple processors;
  - An arbitrary number of computers connected by a network;
  - A combination of both.

What is Parallel Computing?

- The computational problem should be able to:
  - Be broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Be solved in less time with multiple computer resources than with a single computer resource.

Why Use Parallel Computing?

- **Save time and/or money:**
  - In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings.
  - Parallel computers can be built from cheap, commodity components.

Why Use Parallel Computing?

- **Solve larger problems:**
  - Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer,
  - especially given limited computer memory.
  - Web search engines/databases processing millions of transactions per second

Why Use Parallel Computing?

- **Provide concurrency:**
  - A single computer resource can only do one thing at a time.
  - Multiple computing resources can be doing many things simultaneously.
  - For example, the Access Grid (accessgrid.org) provides a global collaboration network where people from around the world can meet and conduct work "virtually".

Why Use Parallel Computing?

- **Use of non-local resources:**
  - Using computer resources on a wide area network, or even the Internet when local computer resources are scarce.
  - For example: SETI@home (setiathome.berkeley.edu) uses 2.9 million computers in 253 countries.
  - Folding@home (folding.stanford.edu) uses over 450,000 cpus globally

Why Use Parallel Computing?

- **Limits to serial computing:**
  - Both physical and practical reasons pose significant constraints to simply building ever faster serial computers:
    - Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware.
      - Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond).
      - Increasing speeds necessitate increasing proximity of processing elements.

Why Use Parallel Computing?

- Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip.
  - However, even with molecular or atomic-level components, a limit will be reached on how small components can be.

Why Use Parallel Computing?

- Economic limitations - it is increasingly expensive to make a single processor faster.
  - Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

Classification of Parallel Machines

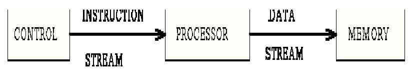
- **Models of Computation (Flynn 1966)**
- Any computer, whether sequential or parallel, operates by executing instructions on data.
- a stream of **instructions** (the algorithm) tells the computer what to do.
- a stream of **data** (the input) is affected by these instructions.

Classification of Parallel Machines

- Depending on whether there is one or several of these streams, we have four classes of computers.
- Single Instruction Stream, Single Data Stream : SISD.
- Multiple Instruction Stream, Single Data Stream : MISD.
- Single Instruction Stream, Multiple Data Stream : SIMD.
- Multiple Instruction Stream, Multiple Data Stream : MIMD.

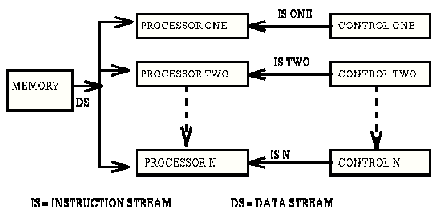
SISD Computers

- This is the standard sequential computer.
- A single processing unit receives a single stream of instructions that operate on a single stream of data.



MISD Computers

- N processors, each with its own control unit, share a common memory.



MISD Computers

- There are N streams of instructions (algorithms / programs) and **one** stream of data.
- Parallelism is achieved by letting the processors do different things at the same time on the same datum.
- MISD machines are useful in computations where the same input is to be subjected to several different operations.

Example

- Checking whether a number Z is prime.
- A simple solution is to try all possible divisions of Z.
- Assume the number of processors, N, is given by N = Z-2.
- All processors take Z as input and tries to divide it by its associated divisor.
- So it is possible in **one step** to check if Z is prime.

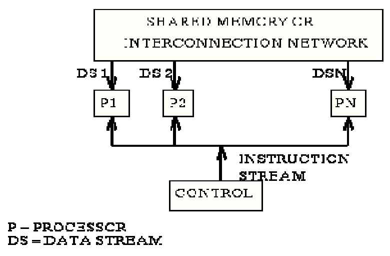
MISD Computers

- More realistically if N < Z-2 then a subset of divisors would be assigned to each processor.
- For most applications MISD are very awkward to use and no commercial machines exist with this design.

SIMD Computers

- All N identical processors operate under the control of a single instruction stream issued by a central control unit.
- (to ease understanding assume that each processor holds the same identical program.)
- There are N data streams, one per processor so different data can be used in each processor.

SIMD Computers



SIMD Computers

- The processors operate **synchronously** and a **global clock** is used to ensure **lockstep** operation.
- i.e. at each step (global clock tick) all processors execute the same instruction, each on a different datum

SIMD Computers

- Array processors such as the ICL DAP (Distributed Array Processor)
- and pipelined vector computers such as the CRAY I & 2 and CYBER 205 fit into the SIMD category.
- SIMD machines are particularly useful to solve problems which have a regular structure. i.e. the same instruction can be applied to subsets of the data.

Example

- Adding two matrices A + B = C.
- Say we have two matrices A and B of order 2 and we have 4 processors.
- A11 + B11 = C11 ...A12 + B12 = C12
- A21 + B21 = C21 ...A22 + B22 = C22
- The same instruction is issued to all 4 processors (add the two numbers) and all processors execute the instructions simultaneously.
- It takes one step as opposed to four steps on a sequential machine.

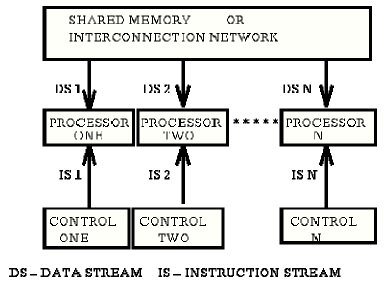
SIMD Computers

- An instruction could be a simple one (eg adding two numbers) or a complex one (eg merging two lists of numbers).
- Similarly the datum may be simple (one number) or complex (several numbers).
- Sometimes it may be necessary to have only a subset of the processors execute an instruction i.e. only some data needs to be operated on for that instruction.

MIMD Computers (multiprocessors /multicomputers)

- This is the most general and most powerful of our classification.
- We have N processors, N streams of instructions and
- N streams of data.

MIMD Computers



MIMD Computers

- Each processor operates under the control of an instruction stream issued by its own control unit.(i.e. each processor is capable of executing its own program on a different data.
- This means that the processors operate **asynchronously** (typically) i.e. can be doing different things on different data at the same time.
- As with SIMD computers communication of data or results between processors can be via a shared memory or interconnection network.

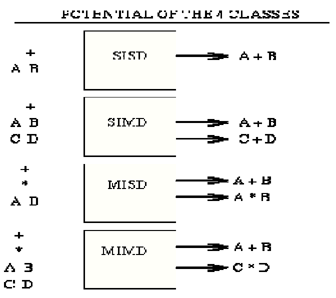
MIMD Computers

- MIMD computers with shared memory are known as **multiprocessors** or **tightly coupled machines**.
- Examples are ENCORE, MULTIMAX, SEQUENT & BALANCE.
- MIMD computers with an interconnection network are known as **multicomputers** or **loosely coupled machines**.
- Examples are INTEL iPSC, NCUBE/7 and transputer networks.

MIMD Computers

- **Note:** Multicomputers are sometimes referred to as distributed systems.
- This is INCORRECT.
- Distributed systems should, for example, refer to a network of personal workstations (such as SUN's) and
- even though the number of processing units can be quite large the communication in such systems is currently too slow to allow close operation on **one** job.

Potential of the 4 classes



Some General Parallel Terminology

- **Supercomputing / High Performance Computing (HPC)**
  - Using the world's fastest and largest computers to solve large problems
- **Node**
  - A standalone "computer in a box". Usually comprised of multiple CPUs/processors/cores.
  - Nodes are networked together to comprise a supercomputer.

Parallel Terminologies

- **CPU / Socket / Processor / Core**
  - In the past, a CPU (Central Processing Unit) was a singular execution component for a computer.
  - Then, multiple CPUs were incorporated into a node.
  - Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit.

Parallel Terminologies

- CPUs with multiple cores are sometimes called "sockets" - vendor dependent.
- The result is a node with multiple CPUs, each containing multiple cores.

Parallel Terminologies

- **Task**
  - A logically discrete section of computational work.
  - A task is typically a program or program-like set of instructions that is executed by a processor.
  - A parallel program consists of multiple tasks running on multiple processors.

Parallel Terminologies

- **Pipelining**
  - Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing.

Parallel Terminologies

- **Shared Memory**
  - From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory.
  - In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.

Parallel Terminologies

- **Symmetric Multi-Processor (SMP)**
  - Hardware architecture where multiple processors share a single address space and access to all resources; shared memory computing.

Parallel Terminologies

- **Distributed Memory**
  - In hardware, refers to network based memory access for physical memory that is not common.
  - As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

Parallel Terminologies

- **Communications**
  - Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

Parallel Terminologies

- **Synchronization**
  - The coordination of parallel tasks in real time, very often associated with communications.
  - Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point.

Parallel Terminologies

  - Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.
- **Granularity**
  - In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.

Parallel Terminologies

  - ***Coarse:*** relatively large amounts of computational work are done between communication events
  - ***Fine:*** relatively small amounts of computational work are done between communication events

Parallel Terminologies

- **Observed Speedup**
  - Observed speedup of a code which has been parallelized, defined as:

```
 wall-clock time of serial execution
 -----------------------------------
wall-clock time of parallel execution
```

- One of the simplest and most widely used indicators for a parallel program's performance.

Parallel Terminologies

- **Parallel Overhead**
  - The amount of time required to coordinate parallel tasks, as opposed to doing useful work.
  - Parallel overhead can include factors such as:
    - Task start-up time
    - Synchronizations
    - Data communications
    - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
    - Task termination time

Parallel Terminologies

- **Massively Parallel**
  - Refers to the hardware that comprises a given parallel system - having many processors.
  - The meaning of "many" keeps increasing, but currently, the largest parallel computers can be comprised of processors numbering in the hundreds of thousands.

Parallel Terminologies

- **Embarrassingly Parallel**
  - Solving many similar, but independent tasks simultaneously; little to no need for coordination between the tasks.
- **Scalability**
  - Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors.

- Factors that contribute to scalability include:
  - Hardware - particularly memory-cpu bandwidths and network communications
  - Application algorithm
  - Parallel overhead related
  - Characteristics of your specific application and coding

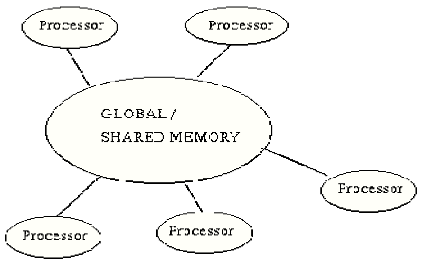
Parallel Computer Memory Architectures

- Shared memory
  - **Uniform Memory Access (UMA)**
  - **Non-Uniform Memory Access (NUMA)**
- Distributed Memory

SHARED MEMORY

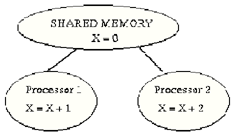
- This consists of a global address space which is accessible by all N processors.
- A processor can communicate to another by writing into the global memory where the second processor can read it.

SHARED MEMORY and SHARED VARIABLES



SHARED MEMORY

- **Shared memory** solves the interprocessor communication problem but introduces the problem of simultaneous accessing of the same location in the memory.
  Consider.



SHARED MEMORY

- i.e. x is a shared variable accessible by P1 and P2. Depending on certain factors, x=1 or x=2 or x=3.
- if P1 executes and completes x=x+1 before P2 reads the value of x from memory then **x=3** similarly if P2 executes and completes x=x+2 before P1 reads the value of x from memory then **x=3**

SHARED MEMORY

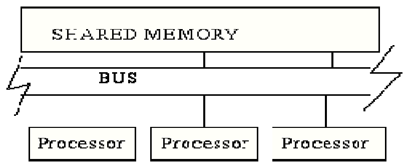
- if P1 and P2 read the value of x before either has updated it then the processor which finishes last will determine the value of x.
- if P1 finishes last the value is **x=1**
- if P2 finishes last the value is **x=2**
- In a multiuser, real time environment the processor which finishes last would vary from run to run - so the final value would vary.

SHARED MEMORY

- Also, even if they finish at the same time only one value of x can be stored in the location for x.
- This gives rise to NON-DETERMINANCY - when a parallel program with the same input data yields different results on different runs.
- Non- determinancy is caused by **race conditions**.
- A race is when two statements in concurrent tasks access the same memory location,
- at least one of which is a write,
- and there is no guaranteed execution ordering between accesses.

SHARED MEMORY

- The problem of non-determinancy would be solved by synchronizing the use of shared data.
- That is; if x=x+1 and x=x+2 were mutually exclusive statements i.e. could not be executed at the same time, then x=3 always.
- Shared memory computers e.g. SEQUENT , ENCORE are often implemented by incorporating a fast bus to connect processors to memory.



- However because the bus has a finite bandwidth i.e. finite amount of data it can carry at any instance, then as the number of processors increase the **contention** for the bus becomes a problem.
- So it is only feasible to allow P processors to access P memory locations simultaneously for relatively small P (< 30)

- Shared memory machines can be divided into two main classes based upon memory access times: ***UMA*** and ***NUMA***.

Uniform Memory Access (UMA):

- Most commonly represented today by Symmetric Multiprocessor (SMP) machines
- Identical processors
- Equal access and access times to memory
- Sometimes called CC-UMA - Cache Coherent UMA.

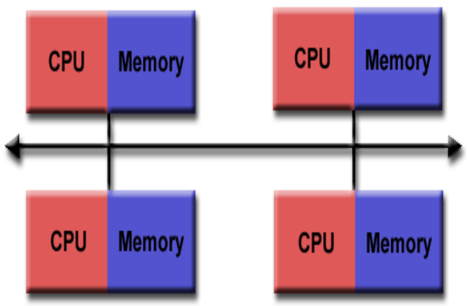
- Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update.
- Cache coherency is accomplished at the hardware level.

Non-Uniform Memory Access (NUMA):

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

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.



- 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.
- The network "fabric" used for data transfer varies widely, though it can can be as simple as Ethernet.