Parallel Computing Platforms

→ Explicit Parallelism needed. Why?

1. Increasing gap in peak & Sustainable Performance (due to memory system performance)

2. Distributed nature of many problems
+
Existing techniques to find automatic parallelism are limited in scope.
[Hard to find at runtime, Need resources, Compile time techniques do not scale very large]

Parallel platforms

Logical
→ Programmer's View

- Control Structure [ways to express Parallel tasks]
- Communication Model [Specifying interaction between tasks]

Physical
→ Actual Hardware Organization

- Architecture [PRAM + Variations]
- Interconnection [Network Topologies]

Logical → Control structure :

→ Parallel tasks can be specified at various levels of granularity.

Program in a set of Programs → to → Instructions within a Program

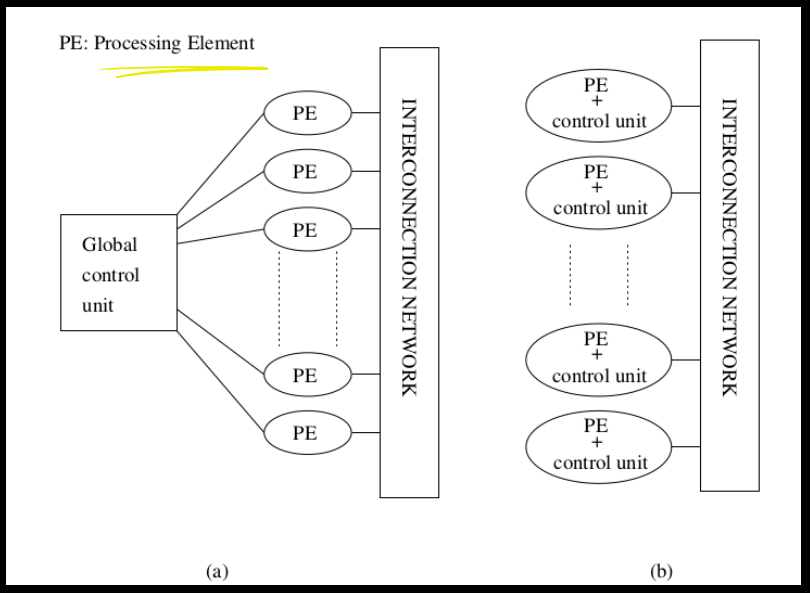
SIMD Vs MIMD

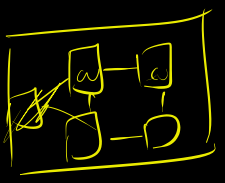
$V_3 = V_1 + V_2$

↓

for(. . . .)

$V_3[i] = V_1[i] + V_2[i]$;





One CU

Synchronous execution of the same instruction on different data

MMX Units in Intel CPU's

Image Processing / Graphics [Structured Computation on arrays]

Conditional execution detrimental to performance

Independent CU's

Capability to execute independent programs

SPMD is a Variant of MIMD (not SIMD)

Any parallel application can be run

No issue about conditional execution

Lesser hardware

Lesser Memory

Specialized hardware leading to design constraints, economic factors, product life-cycle time

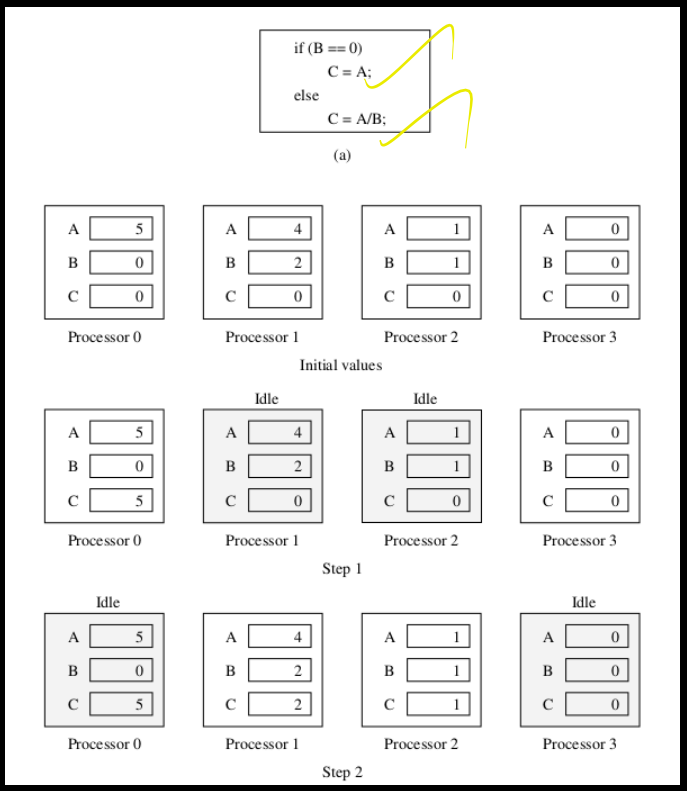
Irregular nature of many applications makes it unpopular

More hardware

More memory for Program & OS at each CPU

General purpose hardware leading to cheap, quick to build using off-the-shelf components

Any application can be run as SPMD



SIMD

MIMD

Logical → Communication Model :

Data exchange

- Shared address Space
- Message Passing

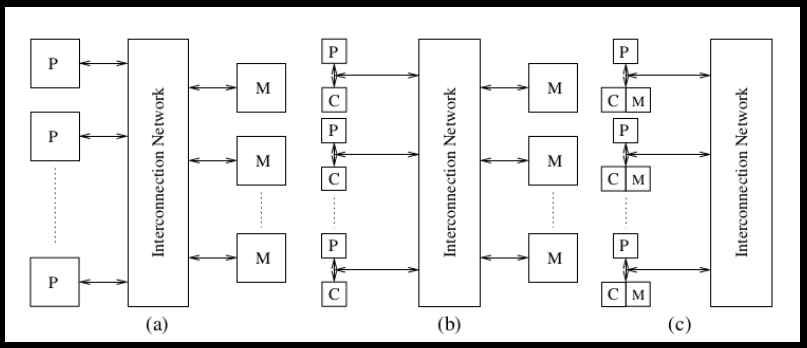
I. Shared Address Space Platforms :

→ Common address space accessible to all processors

→ Processors interact by modifying data objects in shared address space

→ Shared Address space + SPMD = Multiprocessors

→ Memory in shared address space can be local or global



Uniform Memory Access (UMA)

→ Time taken to access any memory word is identical

Non-Uniform Memory Access (NUMA)

Time taken to access certain memory words is longer than others

→ Algorithms to take advantage of locality and thereby structure data & computation accordingly (NUMA)

→ Easy to program model
- Read-only interactions are invisible to programmer as they are exactly same as a serial program
- But Read/Write interactions are harder and need mutual exclusion

→ Cache Coherence issue
- Multiple copies of a single memory word being manipulated by two or more processors at the same time.

↙ Address Translation

↘ Concurrency with well-defined semantics

→ Shared address space Vs Shared Memory Computers

Distributed memory platform (≈ NUMA)
↓
Can present logical view of a shared-address space platform

Shared Memory Computers (≈ UMA)
↓
Can present logical view of a disjoint-address space platform

II. Message Passing Platforms:

Logical View:

- p processing nodes, each with its exclusive address space.
- Interaction between processes running on different nodes by passing messages
- Message passing for data, work & synchronization
- Each node can run a different program
- Cluster of workstations or non-shared-address-space multicomputers
- Can be emulated on a shared-~~address-space~~ memory Computer

Basic Operations:

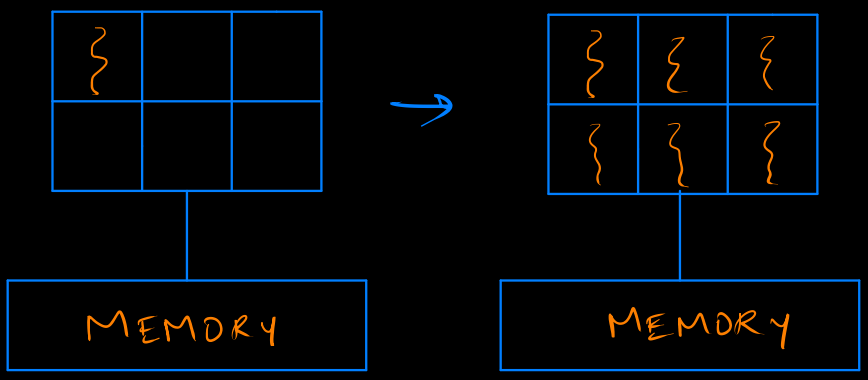
- Get number of processes (numprocs)
- Get self id (whoami)
- Send a message (send)
- receive a message (receive)

Shared address space → pthreads
→ OpenMP

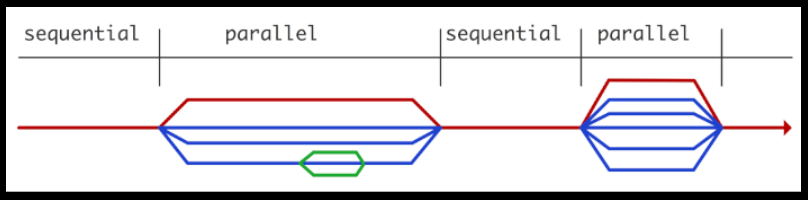
Sai

Basics of OpenMP

→ Threading Model in Shared address space



→ Fork-join model of Parallelism



Master thread forks into multiple threads (copies of master)

Shared + Private data

Each thread has a unique identifier

→ Work sharing constructs divide available Parallelism over threads

→ Can take advantage of multi-core processors