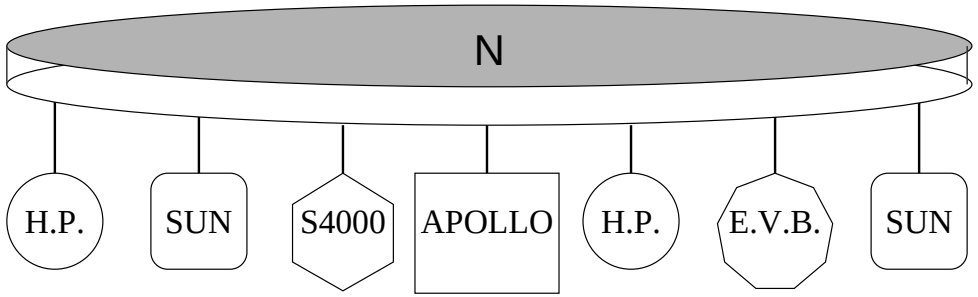
MIMD Parallel Computers

- Heterogeneous MIMD Machines

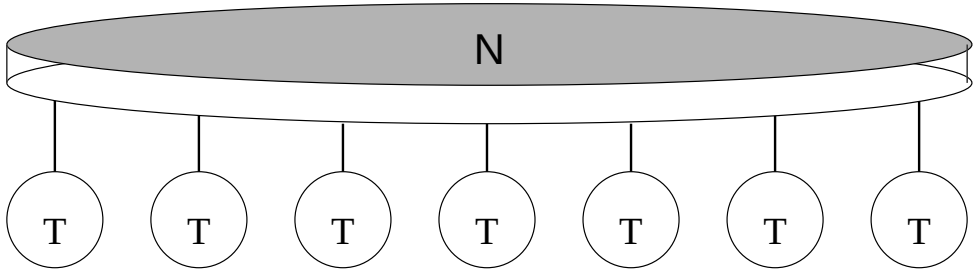
  A MIMD parallel computer constructed of non-identical processing elements.

  (350 various UNIX machines on Ethernet working together to solve a Prolog problem).

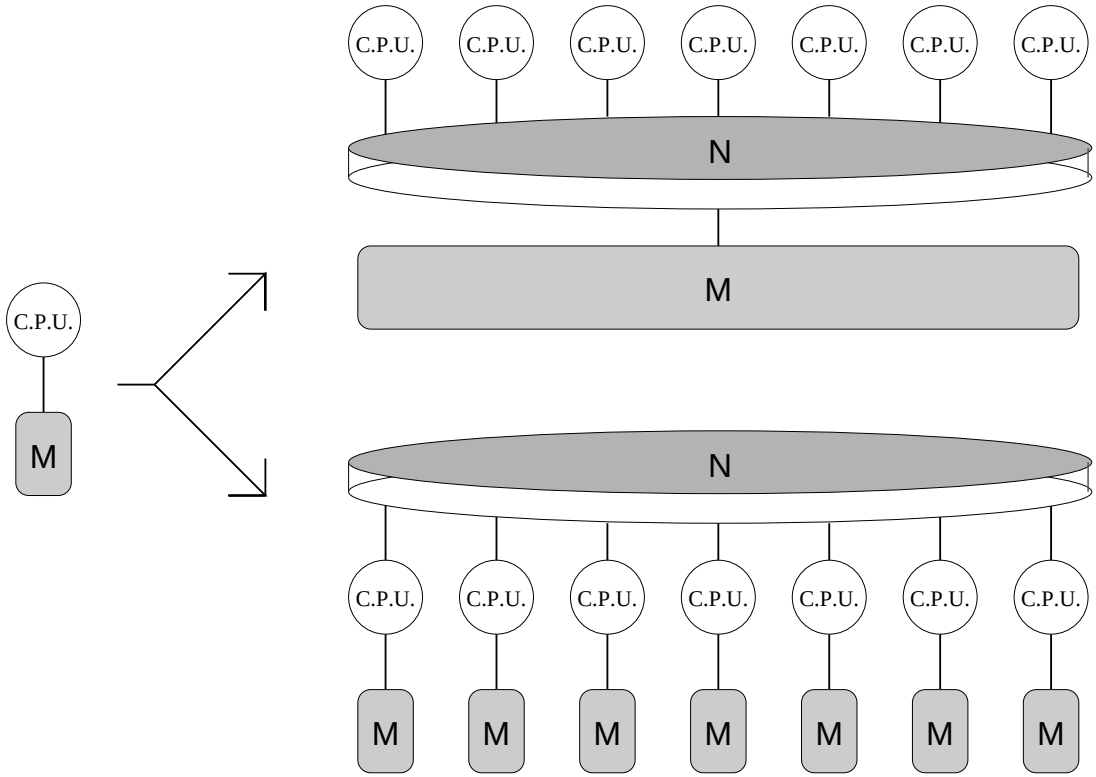


- Homogeneous MIMD Machines (Replicated MIMD Machines)

  A MIMD parallel computer constructed of identical processing elements.



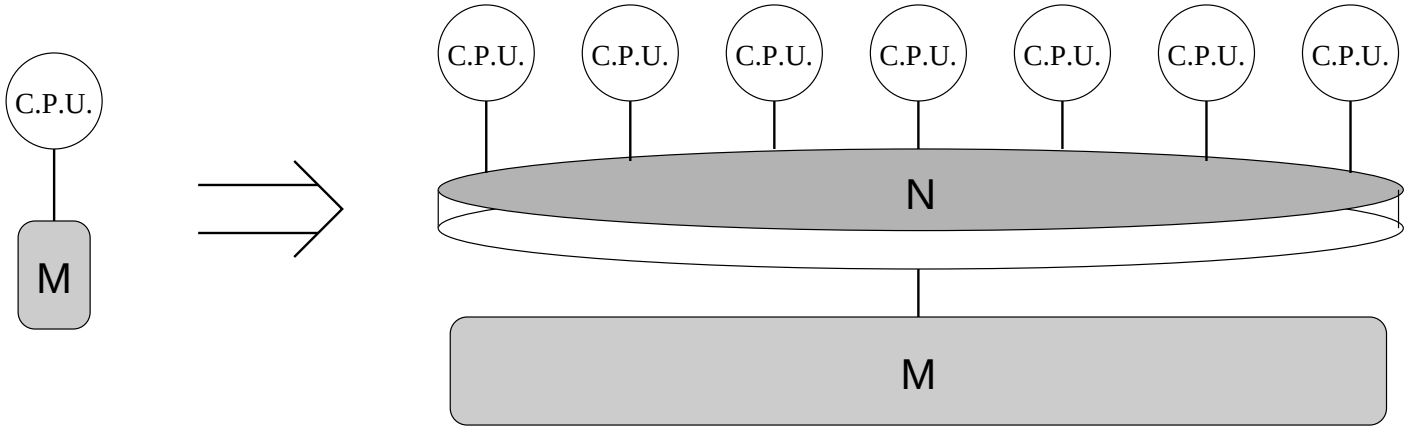
Memory Configuration in MIMD systems



- Shared Memory Multiprocessor Systems
- Distributed Memory Multiprocessor Systems

MIMD - Shared Memory Multiprocessors

Shared Memory Multiprocessor Systems



- Replicated Processing Elements
- Each PE is a full CPU including IPU & ALU [1]
- Single Memory
- Communications Network - Processor to Memory

[1] No extra charge for TLAs

MIMD - Shared Memory Multiprocessors

For years we have multi-tasked on single processors.

We have been giving the user the illusion of one processor per process now we can do it for real!

Unix offers facilities such as -

- *fork()* - forks a new process.
- *wait()* - waits for the new process to die.
- *pipe()* - allows inter-process data transfer.
- *semctl()* - semaphores allow process synchronization.

All communication and synchronization is via shared access to memory.

With a few changes to the operating system we can run existing software with significant speed increase.

Bus Architectures

In a simple system the communications network is a bus.

- Bus Contention

  Multiple requests for bus access.

- Bus Arbitration

  Centralised control over the allocation of bus cycles.

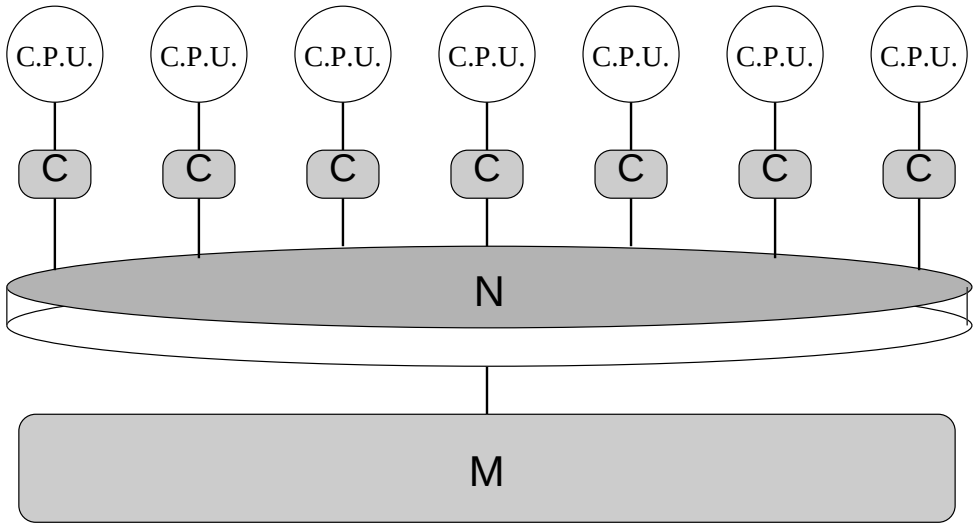
- Delay due to Contention and Arbitration

  Collisions will be more the norm than the exception.

 We have an imbalance between Processing Power and Memory Bandwidth.

MIMD - Shared Memory Multiprocessors

Caches



We can begin to balance Processing Power and Memory Bandwidth by using local caches for each P.E.

Caches

- Reduces Global Memory Access

  Dependent on locality of accesses.

- Cache Coherence

  Must ensure we always access the most up-to-date copy of a data value.

  A write to a location in one cache must result in the update of all copies in all caches, or cause such copies to be discarded from the caches.

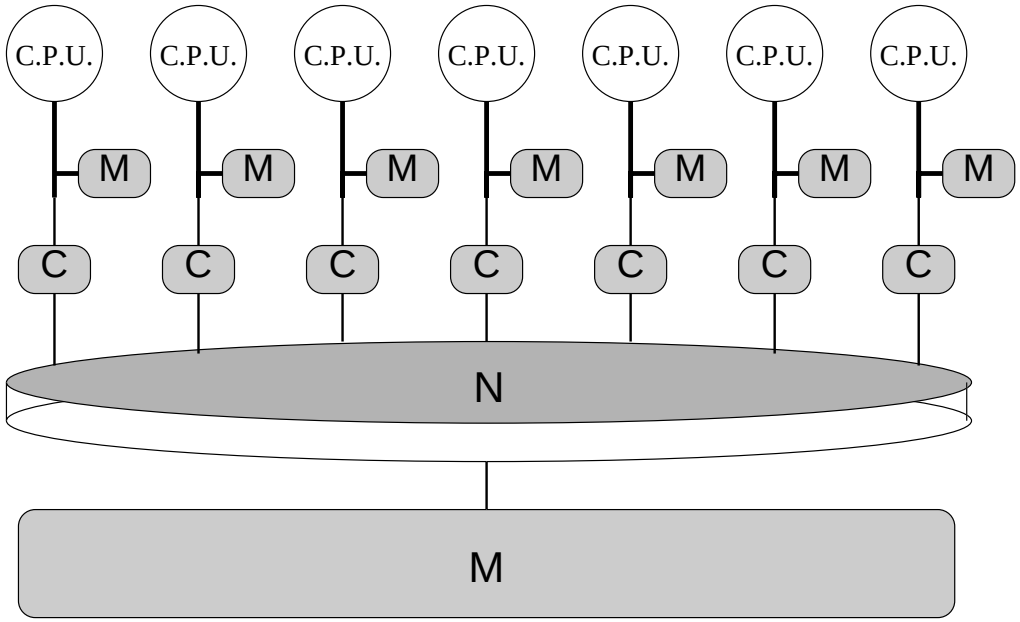
- More Complex System

  More central control. More hardware. More cost.

Our true Memory Bandwidth is now somewhere between the Total Cache Memory Bandwidth and the Global Memory Bandwidth.

MIMD - Shared Memory Multiprocessors

Local Memory

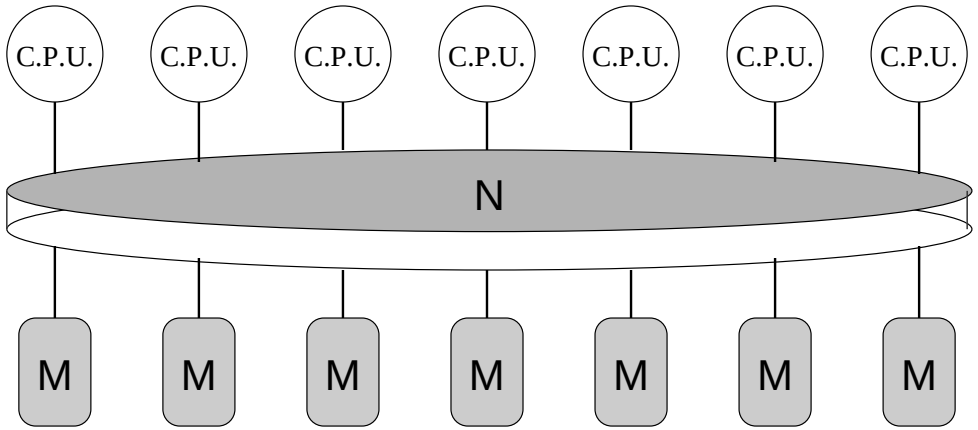


By adding local memory to each PE we can further reduce accesses to global memory.

Local Memory

- Reduces Global Memory Access.
- Deviates from our Shared Memory Ideal.
- To run Efficiently we must modify our programs to use local memory.
- Compiler Technology can help.
- Data which appears local to a process may become global after a process *fork()*.
- More Complex Programming.

Multi Bank Memories



- We allow one memory access per cycle per bank.
- We have a dramatic increase in Memory Bandwidth.

Multi Bank Memories

We still have problems with Contention.

- We have *pathological* programs which require all processors to access the same bank simultaneously.
- Distribute data such that adjacent memory locations are on different banks.

  Use a prime number of banks and a power of two of processors.
- Distribute Randomly

  Use a hashing scheme to determine the location on a data value.

The cost of the system can be very high [2] in order to allow wide bandwidth connection between any processor and any memory bank.

[2] CRAY multiprocessor systems employ this approach.

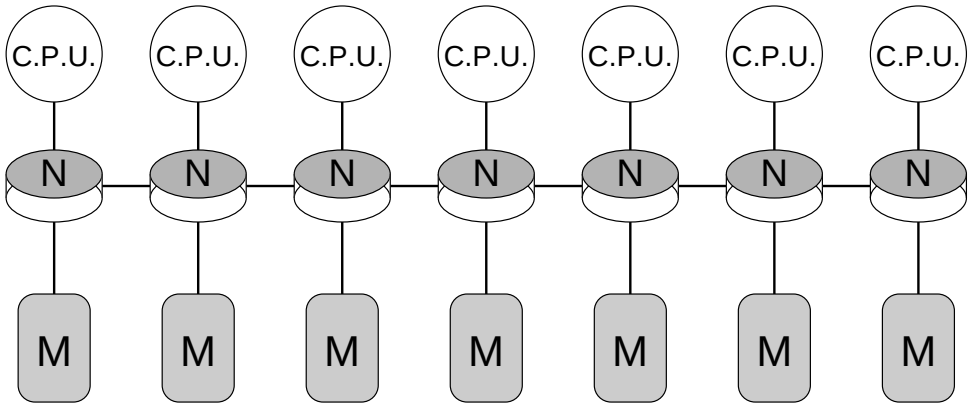
The Limits

The number of processors in a shared memory system is restricted by the following factors.

- Central Control
- Memory Collisions
- Bus Physical Size
- Interconnect Complexity

- With caches and local memory we might manage 32 processors in a shared memory system.
- Not really sensible for Massively Parallel Systems

The Future?



Larger systems are envisaged using distributed communications and multiple memory banks.