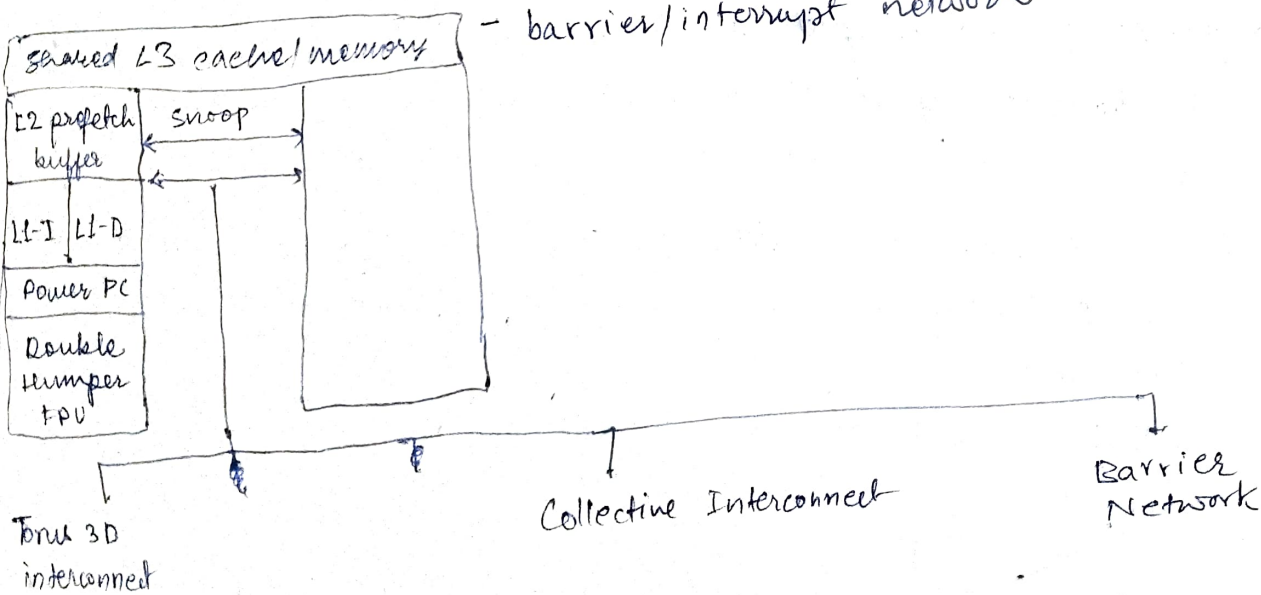
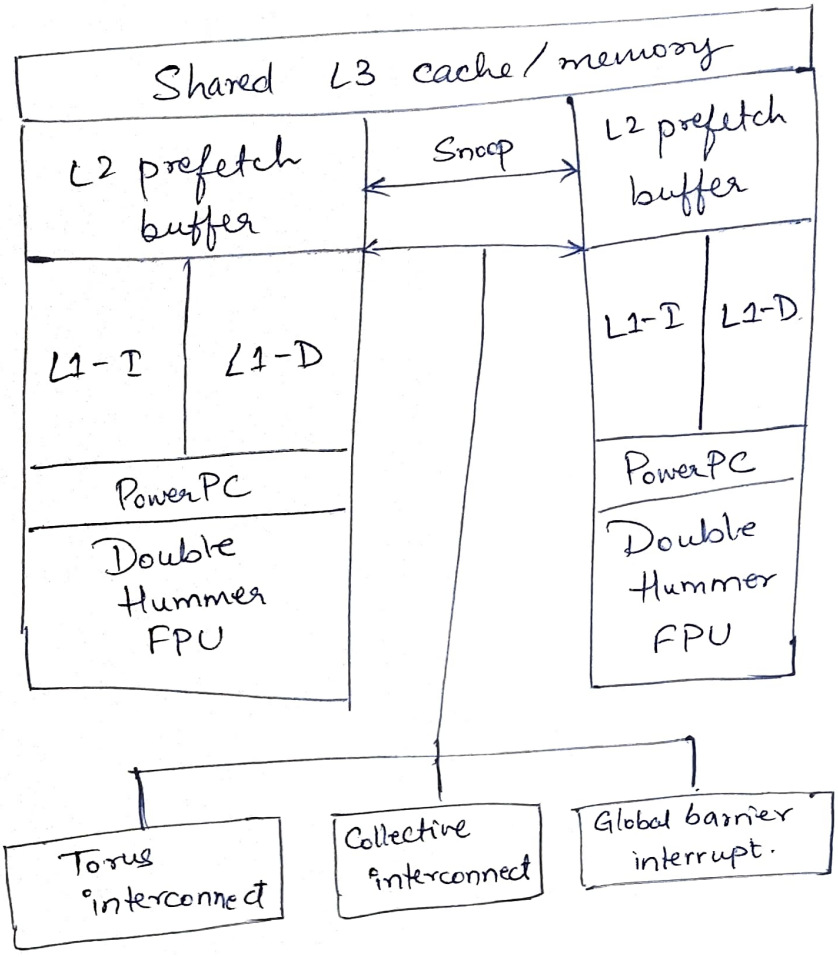
DC

Blue Gene/L

*) super computer
*) processor speed = 770 MHz
*) No. of dual core nodes = 65536 = 2^{16}
*) Each node has: (7 points) :-
  ↳ 32K L1 instructions
  ↳ data caches
  ↳ double hummer optimized floating units
  ↳ 4 MB sequentially consistent shared on chip shared (L3) cache 4MB
  ↳ (6) Bidirectional ports to a [3D] torus interconnect
  ↳ 3 " " to a collective network
  ↳ 4 ports to barrier/interrupt network (चारों ओर से barrier)
*) Different communication channels that exist among processors/nodes are → torus interconnect
  - Collective network
  - barrier/interrupt network



Blue Gene/L - Super Computer.

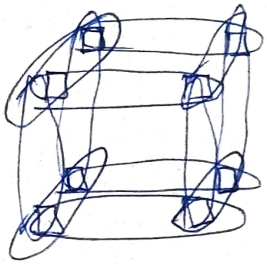


Barrier network

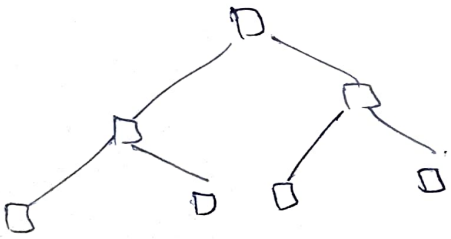
* common wire connecting all nodes.
* for interrupts, barrier synchronization can be done.
* 1.5 μs.

Collective network

3D torus.



* is a mesh with its edges wrapped around.
* connected to 6 nearest neighbor nodes.
* if nodes are not directly connected, data is routed through the edges
* each edge is wrapped around the node
* 6.4 μs



* has arithmetic capabilities,
* Global operations (SUM, MAX, MIN, OR, AND, XOR)
* supports broadcast, simplifying one-to-all communication.
* 10 μs. total

④ What is parallel computing & differences over sequential and distributed.

* The goal of parallel computing has traditionally been to provide performance either in ~~performance~~ terms of processor power or memory; that a single processor cannot provide; thus, the goal is to use multiple processors to solve a single problem.

* Parallel vs sequential Computing

→ Parallel computing involves concurrent computation/simultaneous execution of processess/threads at the same time
Sequential Computing involves a consequtive & ordered execution of processes one after the other.

→ In parallel computing, multiple processes execute at the same time, In seq. comp., computation is modeled after problems with a chronological sequence of events.
chronology samajye

→ In parallel comp, while processes might execute concurrently, yet sub-processes/threads communicate and exchange signals during execution.
In seq. comp, the program executes a process which in turn waits for user input, then another process is executed that processes a return according to user input creating a series of cascading events.

* Parallel vs Distributed Computing

→ many opr. are performed simultaneously
→ single computer required
→ multiple processors perform mul. opr.
→ It may have shared/distributed mem.
→ Processors communicate with each other through bus.
→ improves system performance

→ sys. components are located at diff. locations.
→ multiple computers required.
→ multiple computers perform mul. opr.
→ It has only distributed memory
→ computers communicate through message passing
→ improves system scalability, fault tolerance & resource sharing capabilities

Explain Symmetric Multi Processor Architecture

→ SMP are parallel computers in which all processors access a single logical memory.

~~↳ a portion of the memory~~

→ to achieve consistent memory view, the processors are connected at a common point (memory bus) where each processor can snoop on the ~~mem~~ memory reference activity.

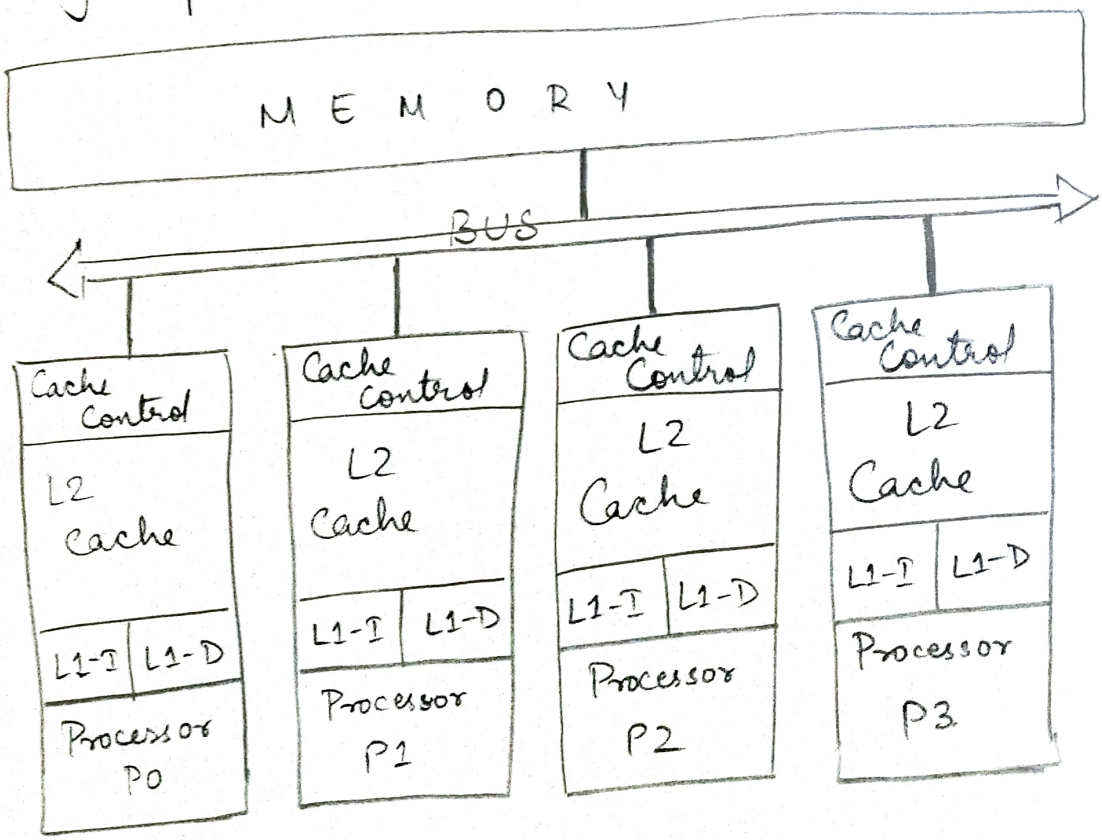
→ The common connection point, bus, is a potential bottleneck as one memory operation takes place at a time.

→ The serial use of the bus limits the no. of processors that can be connected in this way, which implies that SMP's are ~~seen~~ small and have less than 20 connections.

→ Enough L2 caching helps reduce congestion on the bus.

→ SMP's achieve high performance by 2 ways:

1. they tend to be fast as they are small & clustered near the bus.
2. they tend to use shared resources of the bus efficiently by using sophisticated caching protocols.



(6) Describe 6 parallel Computers

Refer notes.

(7) Various Solutions to Counting 3's.

Try 1

```
int *array, len, count;
int count3s()
{ int i;
  count=0;
  for(i=0; i<len; i++)
      if (array[i] == 3)
          count++;
  return count;
}
```

Try 2 Mutex

→ Lock a mutex before incrementing count & unlock the mutex after incrementing.

→ Solution is slower than the sequential code

```
mutex m;
void count3s_thread(int id)
{ int thread_len = len/t;
  int start = id * thread_len;
  for(i= start; i< start + thread_len; i++)
      if (array[i] == 3)
        { mutex_lock(m);
          count++;
          mutex_unlock(m);
        }
}
```

Try 3

→ reduce the no. of critical sections (only 1 per thread)

→ introduce private_count & enter into global count at the end of using mutex.

→ time taken slightly more than serial program.

~~private_600~~

```
private_count [Max_Threads];

mutex m;
void count3s_thread (int id)
{ int thread_len = len/t;
  int start = id * thread_len;
  for (i = start ; i < start + thread_len ; i++)
     if (array[i] == 3)
        private_count [id]++;

  mutex_lock (m);
  count += private_count [id];
  mutex_unlock (m);
}
```

Try 4

* shared L2 cache (P_0 & P_1)
* L2 connected to RAM [no L3 cache]
* add char padding [60]

```
struct padded_int
{ int val
  char pad[60];
} private_count [Max_Threads];
void count3s_thread (int id)
{

  +
  mutex_lock (m);
  count += private_count [id].val;
  mutex_unlock (m);
}
```

(8) Goals of Parallel Programming

There are 2 main goals of parallel programming:

1. Scalability –

→ as the no. of parallel processors increases, physical constraints force design changes that impact program's performance.

→ well written parallel programs can exploit the fast components and avoid overusing the slow components of a parallel computer.

2. Performance Portability –

→ classes of parallel computers –

1. shared memory → multicore processors.
2. shared address space → supercomputers
3. separately addressed memories → clusters.

→ solve portability problem by setting high level of abstraction that none of the differences are visible, then a compiler will map the high level specification to the platform.

→ programs written to port on all classes of computers must be robust to differences in memory structure

* Sources of Overhead

1. Communication (among threads & processes)
2. Synchronization (between threads)
3. Computation (extra; that are not req. in sequential)
4. Memory (needs more memory)

* Threads – is the smallest sequence of programmed instructions that can be managed independently by a scheduler.
Each thread runs its own sequence of instructions.

* Race Condition – occurs when two or more threads can access shared data and they try to change it at the same time..
The thread scheduling algo. can swap b/w threads at any time, therefore, the result of the change in data is dependent on thread scheduling algo, i.e., both threads are racing to access / change the data.

* False Sharing – occurs when processors in a shared memory parallel system make references to different data objects within the same coherence block (cache line or page), thereby inducing "unnecessary" coherence operations.

* Lock Contention – occurs when one thread attempts to acquire a lock held by another process / thread.
The more fine-grained the available locks, the less likely one process / thread will request a lock held by

Mutual Exclusion [Mutex] - is a program object that prevents race condition. It is the requirement that one thread of execution never enters a critical section while a concurrent thread of execution is already accessing critical section

* Memory Reference Mechanisms

① Shared memory
- → simple to design
- → difficult for programmer
- → encourages inefficient programs by making it too easy to make non-local references
- → represents single coherent memory image to multiple threads
- → race condn might occur which leads to difficulty in debugging.

② One sided communication
- → all threads can access all memory spaces but doesn't attempt to keep the memory coherent.
- → removes difficulty/need to implement complex cache coherent protocol
- ⇔ ~~removes difficulty~~
- → synchronization protocols need to be implemented to protect critical variables
- → communication operation can be initiated by only one side.

③ Message Passing
- → no support for shared address space hence local memory used for access.
- → If the data not present in local memory, messages are passed, we send & retrieve data.
- → easier to debug.
- → 2 sided mechanism.