System: Scientific linux.
gcc, g++ compiles.
OpenMP - shared memory system.
www.letshpc.org → Refer this website - generate reports using this.

Performance = ?

algorithms	computer
- They are deterministic	- Performance is non-deterministic
- They are abstract	- not abstract.
- terminatory	- not terminatory

- we cannot isolate system and learn parallel programming.
- our agenda is to solve a ~~prog~~ problem correctly., agenda is accuracy.
- In HPC, we have a change in agenda which is efficiency.

Unit of performance: computations per second. / floating point operations per second: flops/second.

Performance will be written in terms of architecture :-

Performance = frequency of processor × No. of cores × flops/cycle

$$= \frac{\text{cycles}}{\text{second}} \times \frac{\text{flops}}{\text{cycles}} \times \text{No. of cores}$$

$= 2.5$

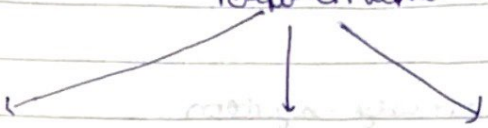
Typical performance = $\frac{2.5 \times 10^9 \text{ cycles}}{\text{second}} \times \frac{4 \text{ flops}}{\text{cycle}}$

= 10 Gflops/second = 10^{10} flops/seconds

Typical ability = 40 Gflops/second.

Page
Date

Performance is decided by 3 important factors.



Architecture algorithm software
parallel environment

- this is all what defines HPC.
- Designing the algorithm is critical,
- architecture needed to optimize the problem, it will give performance to the processor.
- Main problem is to design architecture that fits well with system such as cache.
- code must be scalable and portable.

1. Options, futures and derivatives : J C Hull and Basu : 9th edition (x_{i+1})
2. simulations, By Ross.

For N elements, we have O(N).
Computation = operation + data.
$N = 10^9$.

O(N) → computation complexity is only taken into consideration, the memory complexity is not taken into consideration.

If $N = 10^{20}$, everything does not go into RAM., accessibility has to be taken into consideration.

- unit of operation: flops / second.

Rate = $\frac{10^9 \text{ additions}}{10 \text{ seconds}}$ = 1 Gflops/second.

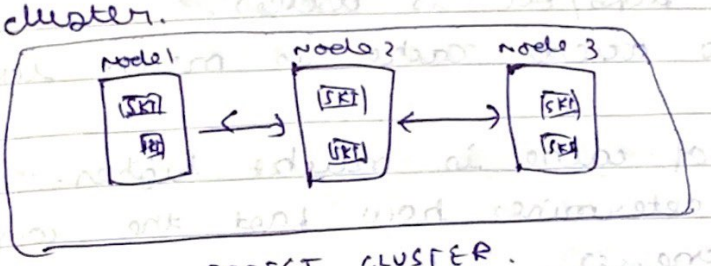
** Theoretical performance = 10 Gflops/second, so try to find a way to reach 10 Gflops/second.

List of things in system architecture:-

1) Architecture - Non deterministic
2) Algorithm design - Deterministic
3) Software environment - Non deterministic

Maximum possible performance out of a particular system for a particular problem.

- Nodes are connected with each other in a cluster.



DAIICT CLUSTER.

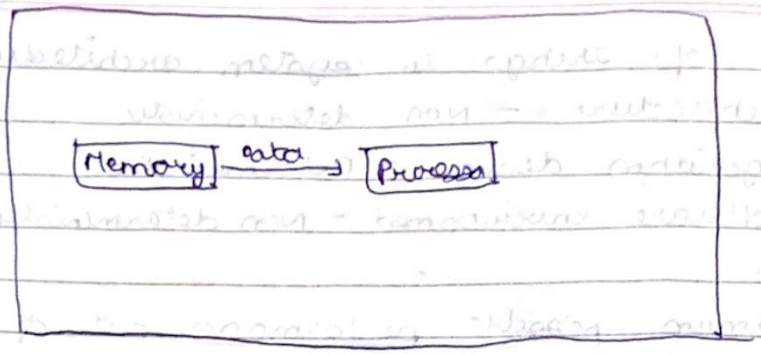
- Inside a node, there is a socket.
- The cores perform flops.
- Inside a socket there is a core

Supercomputer = several nodes.

Node performance = CPU speed × No. of CPU cores × CPU instruction per cycle × No. of CPUs per node.

- Report about hardware, match performance for lab.

- From where performance will come:-
- ~~Device technology~~: cache sits on processor, RAM sits outside processor, so cache is much faster than RAM.



Important factors :-

1) Bandwidth. → MB/sec, GB/sec.

- If bandwidth is not enough, then use of the high flops/sec is useless.

2) Latency to access cache is much lower.

- Bandwidth of cache is much higher.
- All this determines how fast the comparision will be done.
- Write algorithm in such a way that memory complexity is reduced.

Serial :-

- Break a problem into a discrete no of instructions.
- Execute instructions one by one on a single processor.

Parallel :-

- Break problem into discrete processors.
- Each part broken into a series of instruction.
- Each instruction execute simultaneously on several processors.
- Coordination mechanism needed.

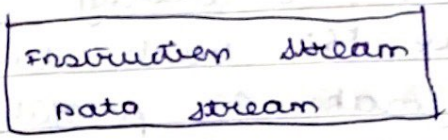
- cannot continue to scale processor frequencies

(no 10 GHz chips), ~~because~~ more powerful will learn processors.

- We cannot increase transistor density

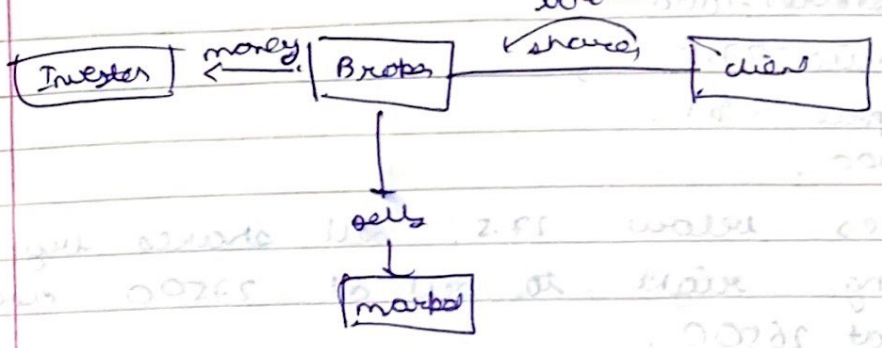
this is why parallel algorithms are needed.

The landscape from Parallel computing Research: A View from Berkeley



observed speedup = $\frac{\text{Serial time}}{\text{Parallel time}}$

- ~~SOR~~ graphs.
- Call option: right to buy.
- Put option: right to sell.
- Buying 100 shares of stock and selling it in London gives $300 profit.
- short selling:- involving an asset not owned.
- Broker



- Investor buys shares, pay an income like interest to broker and then to borrow.
- When investor earns enough money, the money is returned with interest to client,

now investor is owner of the shares.

Investor | Broker | client

- Investor borrows shares from client
- Broker sells and gives money to investor
- Investor buys same volume of shares.
- Gives proceedings to client via broker

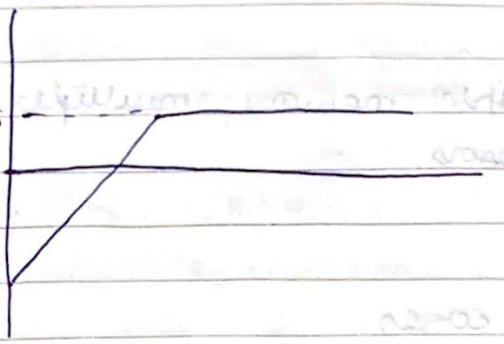
Now, buying / selling of stocks:-

- Buy at $60 after borrowing from bank
- After 1 year, give $63 back to bank and sell at $66 to make $3 profit.
- Now also, sell stock at $60, give in bank buy at $58, earn $63, make $5 profit
- Forward agreement:- agreement to buy or sell an asset at a certain future price at a certain price

Call option: Right to buy at a fixed price
Put option: Right to sell at a fixed price.

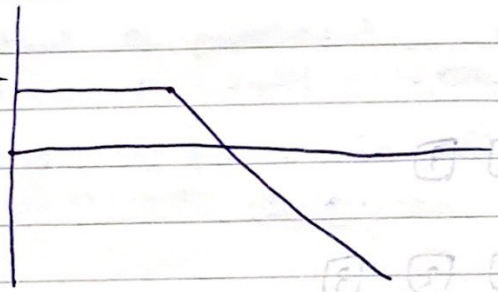
- Price of 1 share = $28.
  No. of shares = 1000.
  Strike price = $27.5,
  Option price = $1,
- cost = $1000.
  If shares below 27.5, sell shares by purchasing right to sell at 27500 and selling at 26500.
- If share above 27.5, no need for put option

27) K=40, ~~loss~~ = 5. option cost = 5.
selling means short.



Loss = 40
Profit = 5.

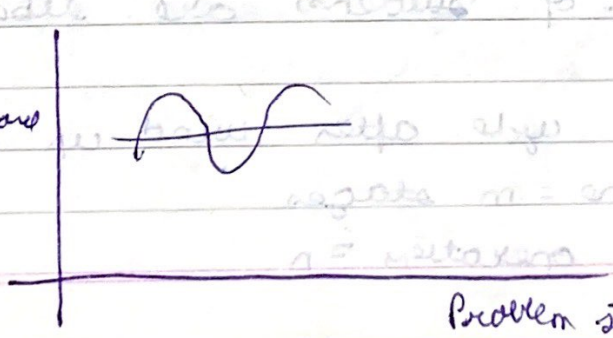
If we have a call option:-



Max profit = 5
Max loss = ∞.

~~28~~ HPC - group of 2.

Peak performance → for benchmarking, setting some standard.
- write a code in such a way so we know about best performance.

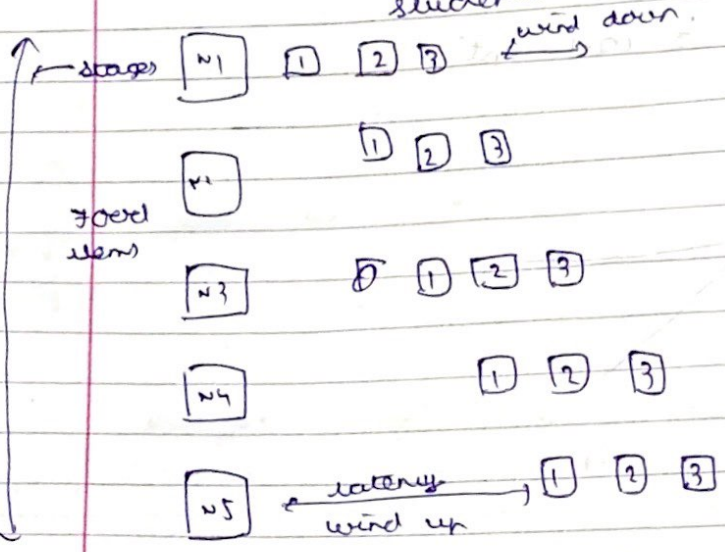


Page
Date

low level benchmarks - programs to understand chief performance characteristics of a system.

* Multiplicity:-
- We have multiple memory, multiple datapaths and multiple processors.

1) Node
2) Threads run on cores.
3) Instructions



- When we complete 5 components, we do an operation
- Ensure that all 5 guys are efficient, take same amount of time.
- One operation per cycle.
- Depth or latency of the pipeline.
- Depth = n, no. of students are like in a loop = n.
- 1 result per cycle after wind up phase.
- Depth of pipeline = n stages
- Total independent operations = n.

Time of pipeline to finish = $m+n-1$

serial time = mn.

$$\text{speedup} = \frac{\text{serial time}}{\text{parallel time}}$$

$$= \frac{nm}{n+m-1}$$

Conclusion: If $n >> m$, we will get good speedup only if loop is large compared to no. of stages. (loop → total no. of instructions).

- We have an optimal no. of stages which has to be figured out.

Throughput of pipeline = $\frac{n}{n+m-1}$

Dividing by n, we get: $\frac{1}{1+\frac{m-1}{n}}$.

The maximum throughput is 1 when $m << n$.

- Von neuman architecture → main memory, CPU, interconnection
- Most important limitation is data access.

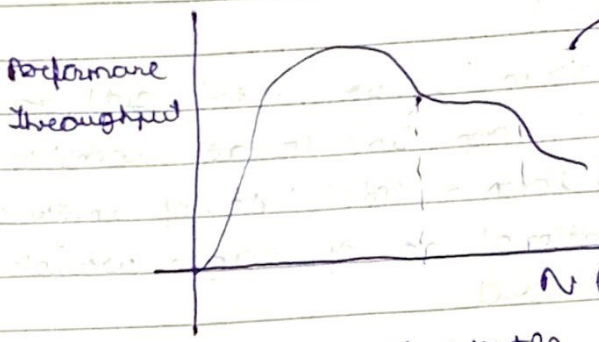
L1 cache = in kilobytes.

L2/L3/main memory = in megabytes.

Latency :- Amount of time required to access 0 bytes of data.

- von neumann bottleneck.

Microbenchmarking:-
- separate influences,
- Determine specific machine capabilities.



$4 \times 1000 \times 4 = 16$ Kbytes.

Machine balance = $\frac{\text{Memory bandwidth GB/sec}}{\text{Peak Performance.}}$

Balance analysis: Theoretical performance of loop based code.

Machine balance ——— code balance.

Machine balance will decrease in future because no. of cores will increase and core performance will increase.

Code balance of loop = $\frac{\text{data traffic}}{\text{No. of floating point operations}}$

Data traffic = no. of variables used.

Computational intensity = $\frac{1}{\text{code balance.}}$

Data → read, store, execution of operation.

Data traffic → performance limiting data path.

Look up: wall clock timer, gettime of day(). function

- Each clock has a resolution
- Be careful about units.
- Plot performance vs no. of loops.
- The graph is used as a low level benchmark.

- a = gettime of day()
- b = get time of day()
- c = a - b // time taken

Principle of locality - spatial and temporal locality.

- Memory access will be on blocks of items instead of individual items, these are called data blocks.
- Cache is k times faster than RAM. (latency + bandwidth).
- Cache reuse ratio = r. (fraction).
- When we are working on data, can we use the data such that we have a, b, c?
- Moment we access something else, understand cache.

  a[i] = b[i] + c[i].

  We need to reuse b[i], c[i], a[i], we may have to reuse cache before throwing it away.
- Fraction of load and store reused from cache.
- Access to main memory time = T_m.
- Cache access time = $\frac{T_m}{k} = T_c$

Average access time = $T_{av} = r T_c + (1-r) T_m$.

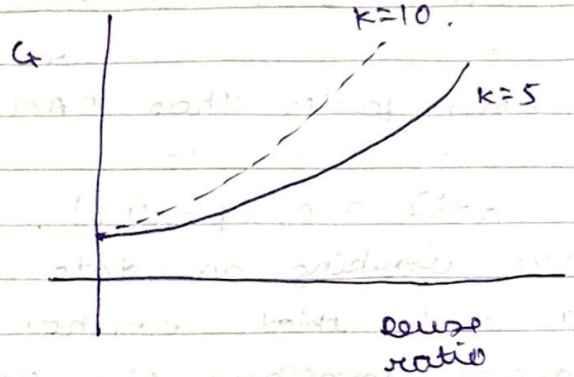
Performance gain, G = $\frac{T_m}{T_{av}}$ (because of cache)

$= \frac{kT_c}{rT_c + (1-r)kT_c}$

Date

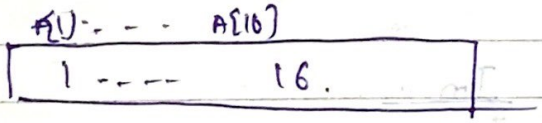
$$G = \frac{k}{r + (1-r)k}$$ → formula for gain.

- If there is huge reuse, i.e. in the case of a fast cache, the gain G will be much faster.
- RAM will sit outside the processor.
- If r=0, no gain.
- Write code in such a way, the reuse rate is greater.



- We called data from RAM to cache, the next time, we keep on ~~do~~ taking data from the cache.
- If no temporal locality, we use spatial locality we do not bring 1 elements, we bring a whole lot of elements together.

A[1] → Not only used, use all elements.



Latency :- The time for 0 bytes of data, latency test is same, so latency per element increases.

- We will increase the cache hit ratio.
- Wherever there is spatial locality, the problem of latency is solved.

cache line length = 16.
spatial locality fixes the hit ratio at ?.
hit ratio = $\frac{15}{16}$ (is the first time, you have to use latency, because nothing was in cache, but for the rest, there is a hit in the cache.

very good ↓
hit ratio.

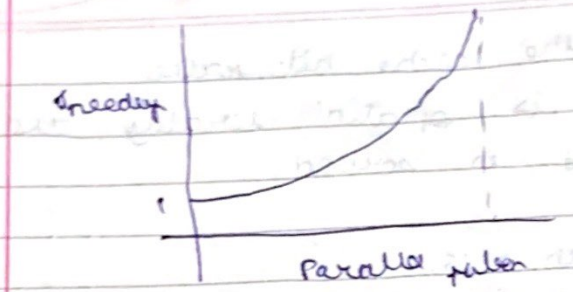
- Parallel algorithm :- No one unique way of doing something.
- Design is on how the data is accessed in cache.
- Efficient parallelization is not parallelizing each step, but to design a new algorithm.

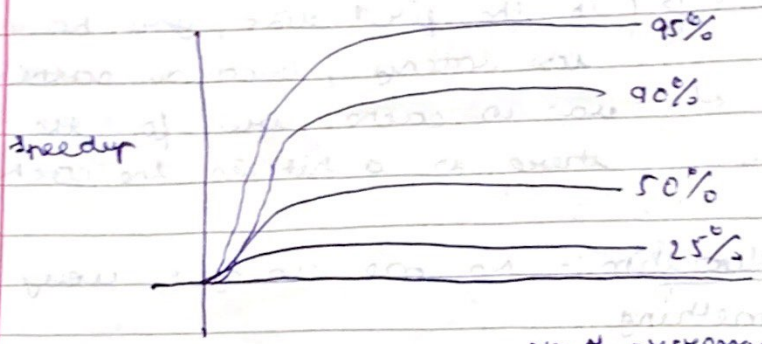
Amdahl's law :-

$$\text{speedup} = \frac{1}{1-p}$$

p = fraction of code that can be parallelized.
p = 0 → no speedup.
If there is dependency, everything cannot be parallel.
p = 0.5, $\text{speedup} = \frac{1}{1-0.5} = \frac{1}{0.5} = 2.$

Theoretical maximum speedup = ∞.





optimizing code:- Performance modelling.

Profiling - serial code

- Find subroutine taking maximum amount of time optimize that subroutine (that particular part of the code.

Compile : f95 -pg file.f

File : gmon.out

Command : gprof gmon.out

a = a + b * * 2.

a = a + b * b.

Both will give the same answer.

Pt * 2.0 → $e^{2 \ln B}$

This optimization technique is called strength reduction.

- A lookup table can be used.

· case A:-

```
do i=1,N
    A(i) = A(i) + s+r*sin(x)
end do
```

case B:-

```
tmp = s+r*sin(x)
do i=1,N
    A(i) = A(i) + tmp
end do
```

Due to cache hit rate, one process will be much faster than the other.

B**2.0 → exp(2 ln B). → very heavy function as more no. of pipeline stages

This is faster.

- Whenever we have a conditional branching, we should try to avoid it.
- The problem is branch miss (like cache miss), if branch miss happens, more time is taken.
- If there is something false, the pipeline has to be flushed, this will take more time, based on a probabilistic model.

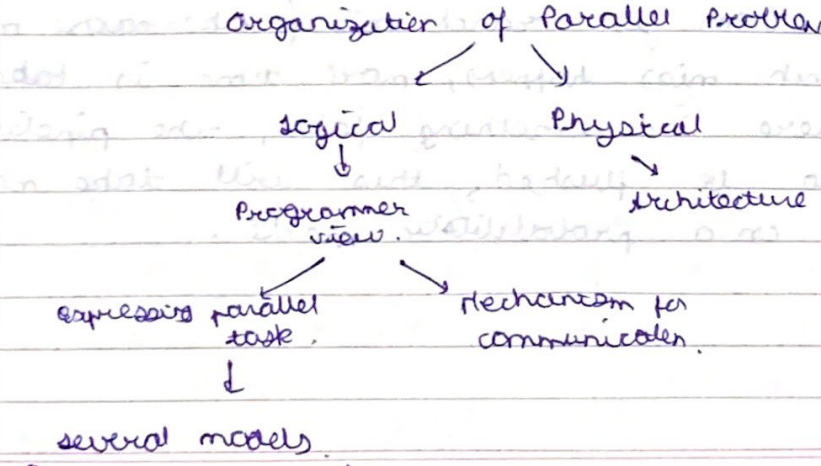
a)
```
do i=1, N, 2
   do j=1, N
      c(i) = c(i) + a(i,j) * b(j)
      c(i+1) = c(i+1) + a(j,i+1) * b(j)
   enddo
enddo
```

b)
```
do i=1, N
   do j=1, N
      c(i) = c(i) + a(j,i) * b(j)
   enddo
enddo
```

In the first case cache hit is taking place, but for the second case, cache hit is not taking place, this is loop unrolling.

Parallel Programming: Scope of parallelism, granularity, locality, load balance, coordination and synchronization.

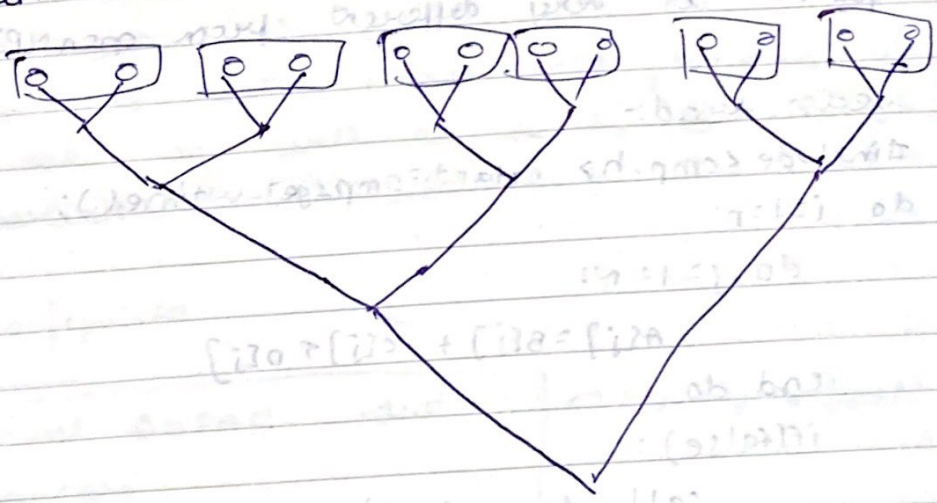
- Parallel programming is harder than sequential programming.

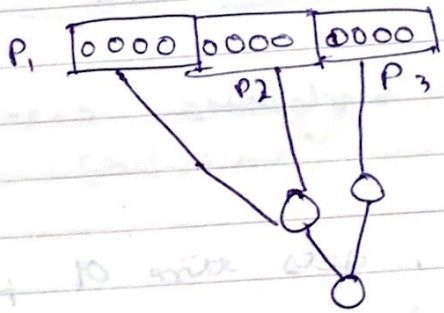


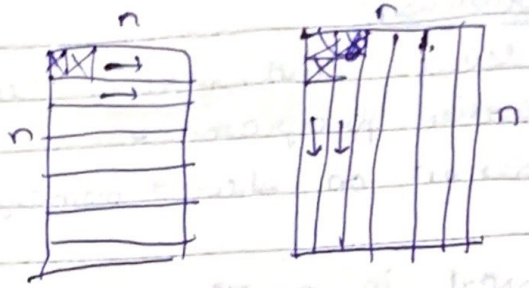
* Shared memory parallelism:-
- A set of compiler directives and library routines for parallel application programmers.
- Performance improvement on shared memory machines.
- use OpenMP
- Multi vender support in openMP
- It is portable.

Granularity: Ratio of computation to communication.
Coarse - large amount of computational work between communication events.

Fine - small amount of computational work between communication events.







we have n ~~processors~~ processes.

4 processors.

$n >> 4$.

finest level of granularity = n^2

highest level of granularity = n.

Logical level will decide level of granularity.

OpenMP is very different from openMPI.

vector triad:-

```
#include <omp.h>  start = omp_get_wtime();
do i=1:r:
    do j=1:N:
        A[j] = B[j] + C[j] * D[j].
    end do.
    if(false):
        call dummy(A,B)
    end do.
end do.
total = omp_get_wtime
```

wtime(): world time, gets time of process, not the ~~system~~.

1) Compile: gcc filename.c -topenmp -o filename-pg.

2) Run: ./filename.out

3) gprof -b filename.out gmon.out

timer function
elapsed time
Resolution of clock

right way: stages are small, and comparable to stages of pipeline.

CPU
↓
Register
L1 cache
L2 cache.

- when L2 cache comes into play, L1 has to take as well as send, not only send, so performance will reduce.

/proc/cpuinfo

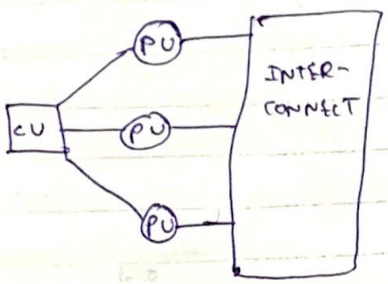
64 bit DDRAM intution → Memory system.
- 8 channels
- 1GHz clock frequency

Access throughput (in bytes/second) = $\frac{8 \text{ bytes}}{\text{transfer}} \times 8 \text{ channels} \times \frac{2 \text{ transfers}}{\text{clock/channel}} \times 10^9 \frac{\text{clocks}}{\text{second}}$

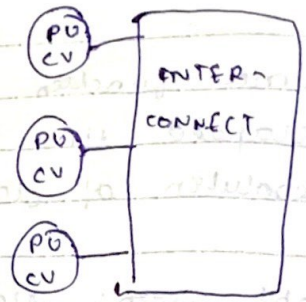
= 128 GB/second

- All threads can access shared cache but not private cache.

SIMD - single instruction multiple data
SPMD - single Program Multiple Data.



SIMD



SPMD

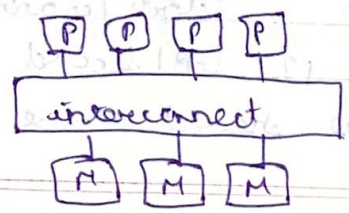
- Control unit gives a single instruction, executed on multiple processes.

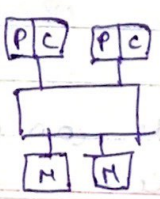
-

- Speed will depend on the program.
- For every clock cycle, one instruction is executed.
- When we have a global memory, it is very easy to read, but very difficult to write, so there comes the concept of private memory.
- In parallel programming, we deal with global and private memory.

UMA - ~~Unified~~ Uniform Memory Access
NUMA - Non Uniform Memory Access.

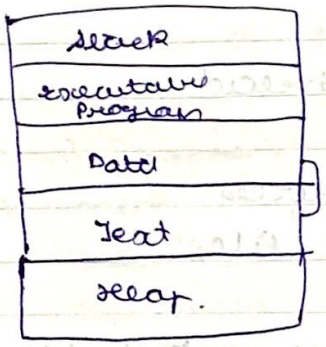
NUMA.

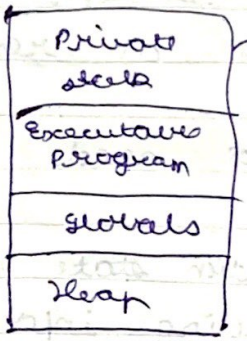




- Each process will take same amount of time, in first diagram, but in the other diagrams, different amount of time is taken.
- Global / private variables.

Process	Thread
- It is a running program.	- It is a lightweight process.
- All processes have a process ID.	
- All processes have a status.	





- Thread is the smallest unit of execution of instructions.
- Process is an execution stream.
  - New
  - Ready
  - Running
  - Waiting
  - Terminated
- Swapping is possible between the intermediate stages.
- Moment a process a thread is created, ID is assigned.

Thread includes :-

1) Thread ID
2) PC
3) Register set (pointer)
4) Stack (pointer)

Shared resources :-

1) Stack session
2) Data session.
3) other OS resources.

The OS needs to keep track of the :-

1) Execution state for each thread,
2) Scheduling information
3) Memory used by the process.
4) Information about open files

- The OS will use a process control block.
- Scheduling has overhead in form of clock cycles elapsed.

SIMD - One control unit gives instruction to multiple data, very easy.

SPMD - scheduling time is more as many control units give multiple instructions.

Mutual exclusion, when we write to a file, we will not allow others to write.

- writing by multiple users is sequential.

reading can be done in parallel.

Dependencies:-

```
for i=1,n
for j=2,m
b[i,j] = ... + b[i,j-1]
```

- for different values of i's we can launch threads, but this is not possible for different values of j as there is a dependency on value of j.

4 Types of dependencies:-
flow, anti, input, output

flow dependence:-

1) statement 'i' precedes 'j' and 'i' captures a value that 'j' uses.

① x=1 ② y=x+2 ③ x=2-w ④ x=y/2.

① → ② Dependencies:-

① → ②

② → ④

2) anti dependence:- ~~after~~

~~statement 'i' precedes 'j'~~

and 'i' captures a value that 'j' computes

① x=1 ② y=x+2 ③ x=2-w ④ x=y/w.

Dependencies:-

② → ③

② → ④

3) Output dependency :-

statement 'i' precedes 'j'
and 'i' computes a value that 'j' also computes.

① $x = 1$ ② $y = x + 2$ ③ $y = 2 - w$ ④ $x = y/2$.

Dependencies:
① → ③
① → ④
③ → ④

4) Input dependency :-

statement 'i' precedes 'j'
and 'i' uses a value that 'j' also uses.

Dependencies :-
① $x = 1$ ② $y = x + 2$ ③ $x = 2 - w$ ④ $x = y/2$.

Dependencies :- (Input dependency not a problem)
③ → ④.

- The most challenging problem is how to remove the dependencies.
- It is very difficult do remove true dependency.

Dependency flows from 'i' to 'j'
↓ ↓
source sink.

- We can have data and task dependency.
use DAG, directed graph.