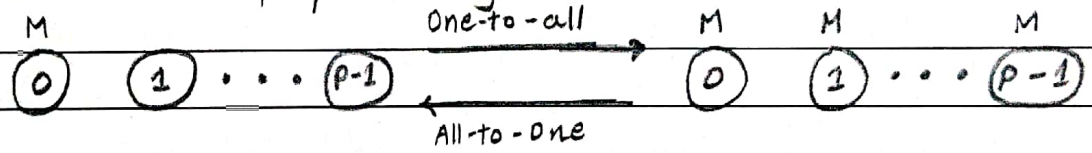
MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

UNIT 3: PARALLEL COMUNICATION

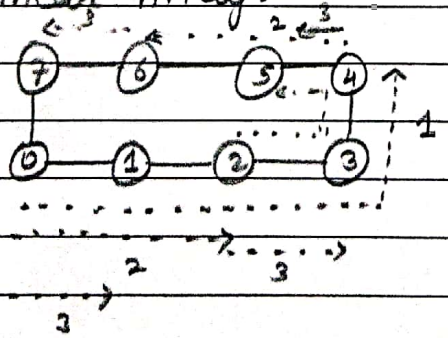
* One to All Broadcast :-
- Single processor sends identical data to all processors.
- Mostly used in parallel operations / algorithm.
- If 'p' is the number of processors and 'm' is the size of data then total 'mp' memory space will be used considering all processors.
- Each processor will have its own copy of data.

* All to one reduction :-
- Data from al processes are combined at single destination process.
- Operations like sum, maximum, minimum, product for a set of numbers can be performed by all to one reduction.



~ These are 3 topologies for the above 2 types of broadcasting :-
i) Ring / Linear Array [1D]. (ii) Mesh . (iii) Hypercube.

• Ring / Linear Array :-



Recursive doubling / One to all broadcast on an eight node ring.

1. Sequential passing of a message from one process to another

causes congestion & is time consuming.

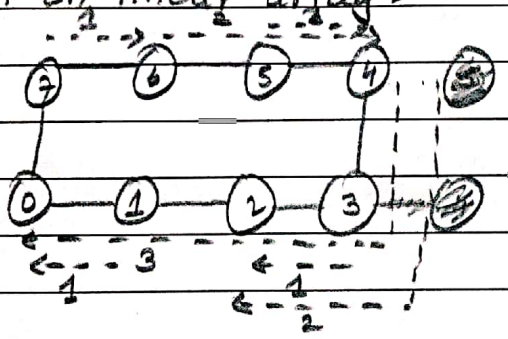
2. To overcome this problem in 1D array we use recursive doubling.

3. For sequential passing message passing takes $O(p-1)$ time whereas recursive doubling can do it in $O(\log p)$.

4. In recursive doubling given we start from 0^{th} node and we want to send message to all $p-1$ nodes then we first send message from 0 to $p/2^{th}$ node.

5. Now both 0^{th} and $p/2^{th}$ node will send message simultaneously in recursive manner.

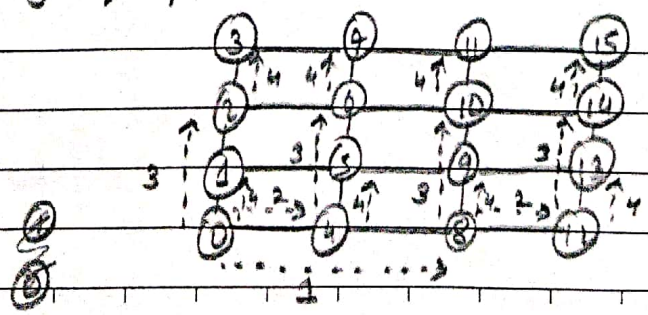
* Reduction on linear array :



- In reduction first all odd processors pass message to the next even processor;
- Then all even processor pass message to even processor.
- Finally $p/2^{th}$ node will pass message to destination node.

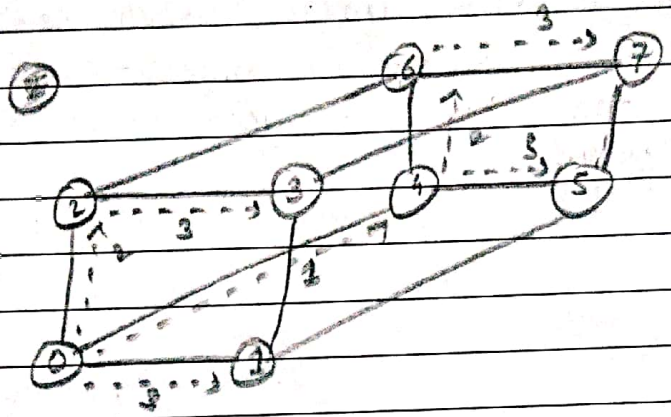
* Mesh :

- Each row & column of a square mesh p can be considered as linear array of $\sqrt{p}$ nodes.



- communication works in two phases: first row-wise and then column-wise.

* Hypercube :

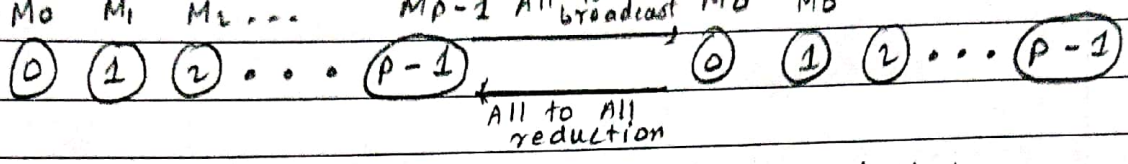


- Similar to 2D mesh, 3D mesh works in two phases.
- A hypercube with 2^d nodes can be considered a d-dimensional mesh.
- We carry out d steps, one in each dimension.
- Communication starts from highest dimension bit.
  ∴ 0 → 4 i.e 000 to 100

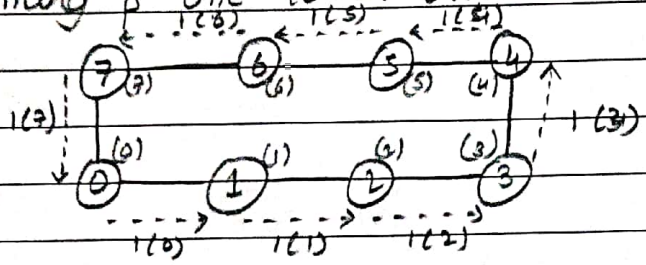
* cost Analysis.
- For p processes, each message will be transfered at least log p times.
- Each tranfer will have cost : ts + tw · m
- Total time taken : $T = (t_s + t_w \cdot m) \cdot \log p$.

* All-to-All Broadcast & Reduction:-
- In all-to-all all p nodes broadcast the message simultaneously.
- All processes may not contain the same message.
- In all to all reduction takes place on every node.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI.
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

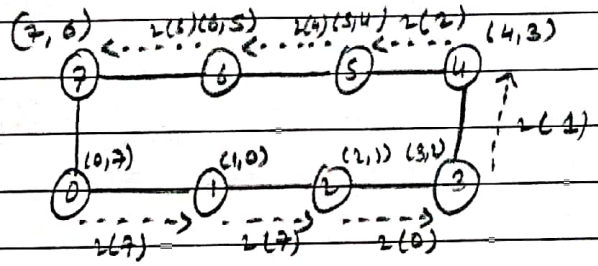


- One approach to perform A-2-A broadcast is by performing p One-to-all broadcast.

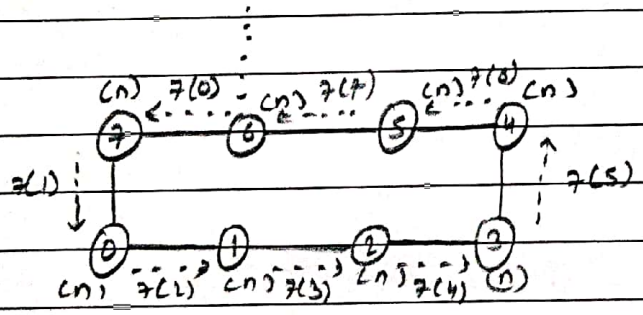


n (m)
↓ ↓
Item Message

1st communication step



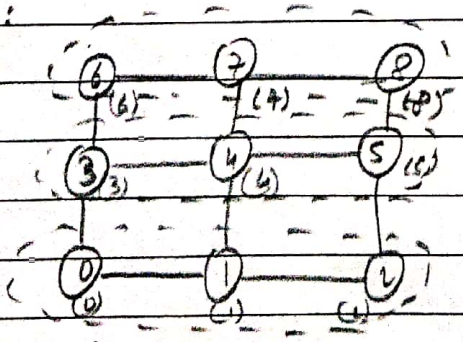
2nd step



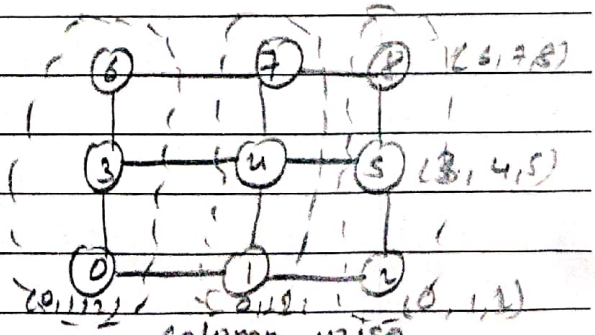
$A = \{1, 2, \ldots$
$n = \{R,$ where $R = 0 \le R \le p-1$
and $R \neq (n+1)\%n$

- For linear or ring it takes 7 rounds.

- Mesh:



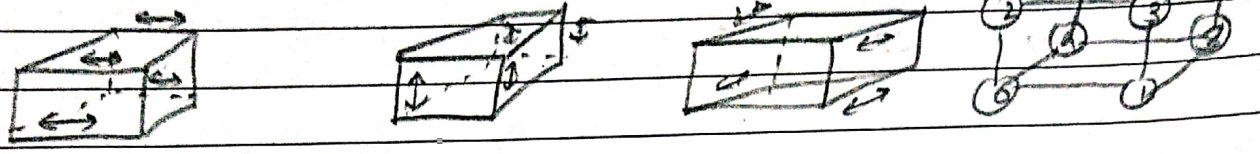
Row-wise distribution



Column-wise distribution.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

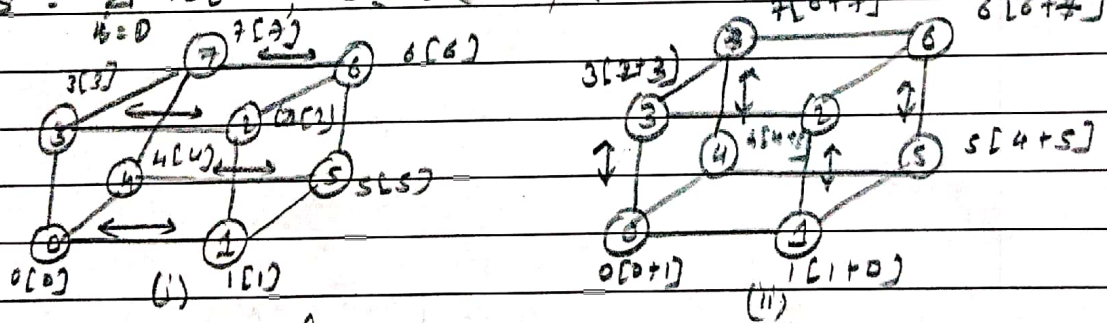
- Hypercube:
- Same here just 3rd step which is depth comes



* Cost Analysis:-
  - $T = 2 t_s(\sqrt{p} - 1) + t_w m(p-1)$ for Mesh & Ring
  - $T = t_s \log p + t_w m(p-1)$ for 3D

- All-reduce & Pre-fix sum operations:-
- All-reduce = All-to-one reduction + All-to-one broadcast.
- After operation identical buffer size m are formed by combining mp buffers.
- Prefix sum:-

$$S = \sum_{k=0}^{k} n_i \quad 0 \le i \le k, \; 1 \le n \le p$$

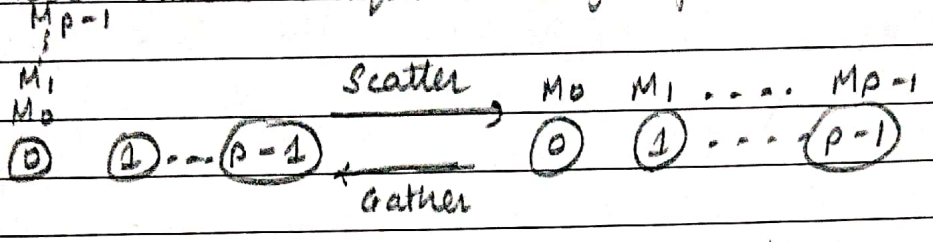


Do same for column (iii)

Final prefix sum (iv)

- Give example.
- Keywords :- Memory - Buffer, process - node.

* Communication using MPI : Scatter & Gather
- One-to-all sends same data to all p nodes.
- In scatter, single node sends unique message of size 'm', to every other node.
- Scatter AKA one-to-all personalized communication.
- Gather operation is dual of scatter.
- Single node collects unique message from each node.



- Gather differs from all-to-one reduce as it does not reduce or combine collected data.

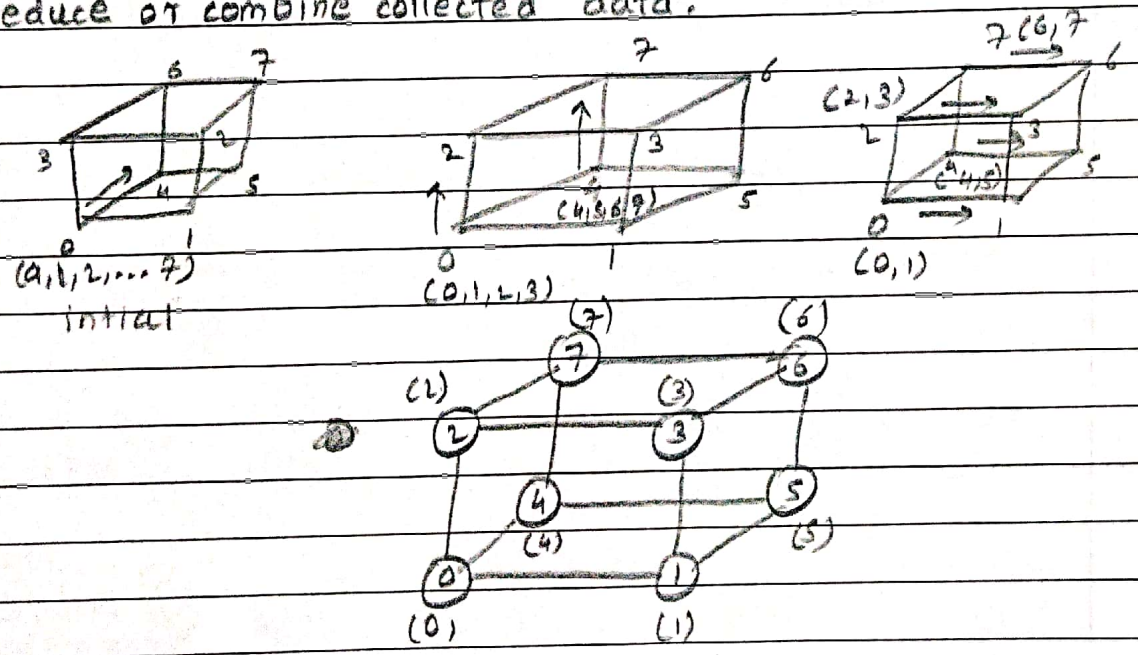


Fig : Scatter operation

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

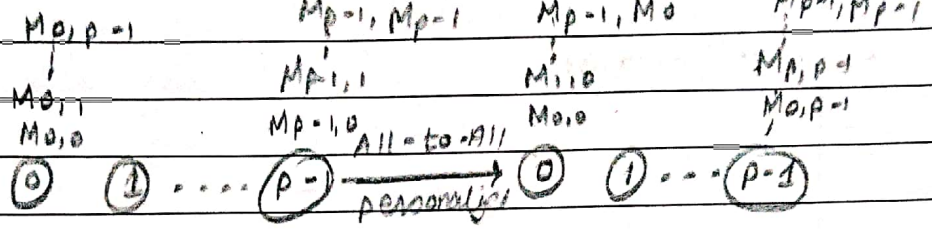
Page No.

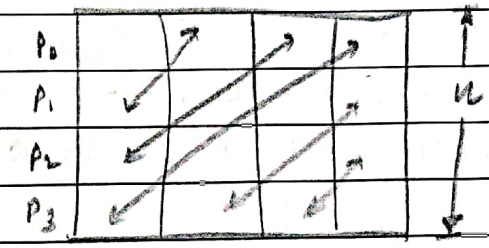
Cost: $T = t_s \log p + t_w m(p-1)$

Same time for 2D & linear.

* All-to-All personalized communication:

- Every node sends a distinct message of size m to every other node.





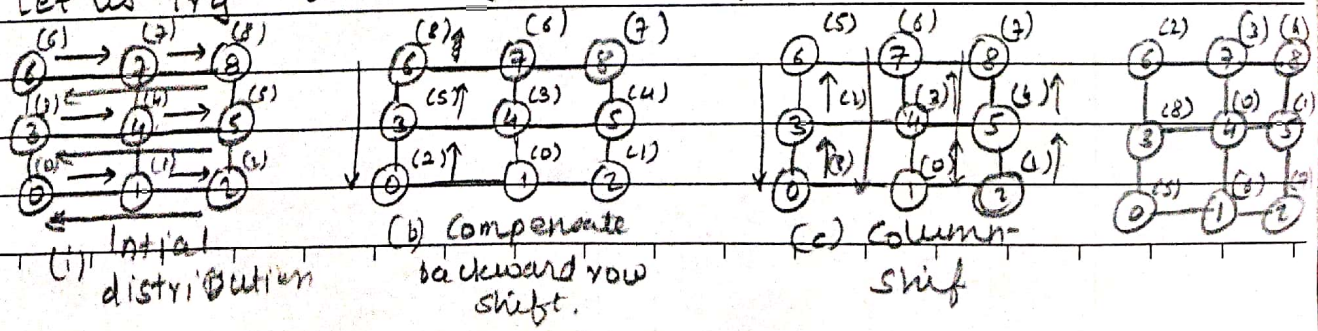
All-to-All personalized communication in a 4×4 matrix.

- It is similar to taking transpose of a matrix.

* Circular Shifts:-

- Used in matrix computation & string/image matching operations.
- It is a broader class of global communication model.
- In a circular q shift, node i sends data to $(i+q) \% p$ in a group of p nodes where $0 < q < p$.
- This is known as permutation.
- Algorithm:- 1) Circular shift row wise, 2) Circular shift only 1st row, 3) Circular shift all column. 4) Done.

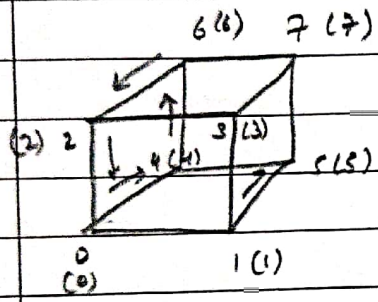
Let us try 3×3 mesh with $q = 4$.



Cost: $T = (t_s + m t_w)(\sqrt{p} + 1)$

- $i = (i + q) \% p$

- Hypercube:
- For circular shift in hypercube values of i^{th} node are assigned to j^{th} node where j is binary reflected gray code.
- Two phases:
  - First phase: 2×4 shift
  - Second phase: 1 shift.



UNIT 4: ANALYTICAL MODELING OF PARALLEL PROGRAMS

- Time Complexity of parallel algorithm depends upon :-
i) Input Size. (ii) No. of processing units (iii) Communication speed of processors. (iv) Relative computation of processors.

* Sources of overhead in parallel programs :-
- Doubling hardware resources may not double the performance.
- Over Typical execution involves Essential computation (Serial), Excess computation (Parallel), Idling, interprocess communication :-
1) Interprocess communication. 2) Idling (Load imbalance, Synchronization)
3) Excess computation (Difference in computation by parallel program and serial program).

* Performance Measure & Analysis :-
- Some metrics are :- 1) Efficiency. 2) Speedup. 3) Cost. 4) Parallel Overhead. 5) Execution Time.

• Speedup :-
- Captures relative benefit of solving a problem in parallel.
- Given as :

$$S = \text{Speedup} = \frac{T_s}{T_p}$$

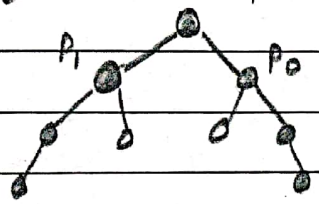
[T_s = Time taken by best sequential algorithm / Time taken by single processor]
[T_p = Time taken by p processors]

- Consider summation of n elements using n processing units.
- Each processor is assigned a number to be added and in end a single processor will hold total sum.
- Total addition will take place in log n steps.
- Addition takes constant time as well interprocess communication.

$\therefore T_p = \theta[\log n] \qquad S = \theta\left[\frac{n}{\log n}\right]$

Speedup.

- Theoretically, speedup cannot be > p.
- Speed greater than p is possible only if each processing element speeds time < T_s/p.
- In practice speedup > p is observed, this is superlinearity effect.
- Happens when serial does more work than parallel.
- Two reasons :- 1) Superlinearity due to caches [Fast Access].
2) Superlinearity due to exploratory decomposition [If working with tree] :



Speedup = $\frac{9}{4}$ = 2.5

* Efficency :- Ratio of Speedup to no. of processing elements.
$E = \frac{S}{P}$ Ideally : $S = p$ and $E = 1$.
In practice $S < p$ and $E < 1$.
It is the measure of fraction of time for which a processor is fully employed.
- Cost :- Product of parrallel runtime & no. of processors used.
- Cost given as : $W = T_p * p$.
- A parallel system is cost optimal if it has same asymptotic growth as the best known sequential algorithm.
- Execution Time :-
- Time elasped from the moment parallel computation starts to the moment it ends.
- Serial runtime is T_s and parallel is T_p.

* Amdahl's Laws :-
- Consider a fixed computational workload W and a number of processors p.
- Let execution rate of i processors be R_i so for

sequential processors ($i = 1$) $R_i = 1$ & $R_n = n$

- Assume sequential workload is αW ($0 \le \alpha \le 1$) and parallel workload is $(1-\alpha)W$.
  $\therefore W_1 = \alpha W$ & $W_n = (1-\alpha)W$.
- Total Time execution is :

$$T_n = \frac{W_1}{R_1} + \frac{W_n}{R_n} = \alpha W + \frac{(1-\alpha)W}{n}$$

$$T_n = W\left[\alpha + \frac{1-\alpha}{n}\right] = W\left[\frac{\alpha(n-1)+1}{n}\right]$$

- Speed up is :-

$$S_n = \frac{T_1}{T_n} = \frac{W/1}{W\left[\frac{\alpha(n-1)+1}{n}\right]} = \frac{n}{1+(n-1)\alpha}$$

This is Amdahl's law, when $n \to \infty$

$$S_\infty = \frac{1}{\alpha}$$

Even when $n \to \infty$, Speedup cannot be increased beyond $(1/\infty)$ this is a bottleneck for multiprocessors.

* Gustafson's law :-

- Consider Workload is increased to n-node machine,
  $\therefore W' = \alpha W + (1-\alpha) n W$
- Speed up is :-

$$S' = \frac{\text{Single processor execution}}{\text{Parallel process execution}}$$

$$S' = \frac{(\alpha W + (1-\alpha) n W)/1}{\frac{\alpha W}{n} + \frac{(1-\alpha)\alpha W}{\alpha}} \quad \ldots \left[T = \frac{W}{f} \text{ for } i = 1,\ T = \frac{W}{1} = W\right]$$

$$S' = \frac{\alpha + n - n\alpha}{\frac{\alpha}{n} + (1-\alpha)} = \frac{(\alpha + n - n\alpha)n}{(\alpha + n - n\alpha)}$$

$S' = \frac{\alpha + (1-\alpha)n}{\frac{\alpha}{n} + (1-\alpha)} = \frac{\alpha n + (1-\alpha)n^2}{\alpha + n(1-\alpha)}$

MANJARA CHARITABLE TRUST

RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI

JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

It simplifies as $S'n = \alpha + (1-\alpha)n$

- So Gustafson's law says that true parallel power is only achieveable when large parallel problem is applied.

* Effect of Granularity on performance.
- No. of task divisions $\propto$ Granularity.
- Fine granularity :- Decomposition into large no. of tasks.
- Coarse granularity :- ——————— " — small — " — " —
- Degree of concurrency $\propto$ Granularity.
- In practice we do fine granularity.
- Using less processing elements than available is called scaling down of a parallel system.
- If there are n inputs and p processing elements and $p < n$ then we can design a parallel algo that works for n processing elements by using n/p virtual processing elements.
- As processing element decrease by n/p the workload at each processing unit increases by a factor of n/p.
- Granularity impacts :-

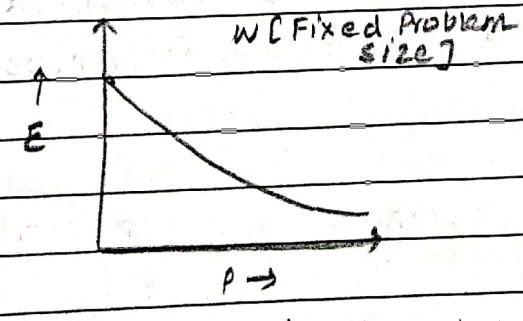
1) It affects performance of parallel computers.
2) Finer granularity increases speedup.
3) However synchronization overload negatively affects performance.

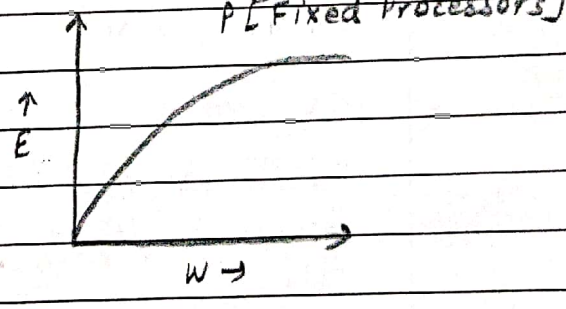
* Scalability of parallel systems :-
- Capacity to increase speedup in proportion to number of processing elements.
- Reflects parallel system's ability to utilize increasing processing resources effectively.
- Efficency is given as :-

$$E = \frac{S}{P} = \frac{T_S}{p\,T_p} = \frac{1}{1 + \frac{T_o}{T_S}}$$

- Total overhead To is an increasing f^n of p.
- Every parallel program always has a sequential that runs for time ts.
- ∴ Total overhead = $(p-1) \times ts$
- However overhead increases linearly with p, however, it may increases superlinearly due to communication overhead, idealling, excess computation.
- ∴ As p ↑ E ↓ goes down $E \propto 1/p$.
- Scale more then efficency shall decrease.
- Efficency may increase if problem size is increased keeping number of processors constant.

- Isoefficency metric of scalability :-
- Helps determine best algorithm/architecture combination for a particular problem without trying them all.
-





- Isoefficency function helps determine ease with which a parallel system can maintain constant efficency.
- A parallel system is cost-optimal iff
  $p\,Tp = \theta(W)$ where W is ~~overhead~~ Overhead.
  And $W = \Omega(To(W,p))$
  ∴ Parallel system is cost-optimal if its overhead function does not exceed its problem size asymtotically.

- W(p) is asymptotic lower bound on isoefficency.
- Isoefficency fn is optimal only if degree of concurrency is $\theta(W)$.

* Minimum execution time & Minimum cost :-

- As the number of processors increase the parallel runtime continues to decrease and asymptotically approaches a minimum value, or goes on increasing after a min val.
- We give min execution times as :-

$$\frac{d}{dp} T_p^{min} = 0$$

This gives the no. of processing elements for which Tp is min.

- However max. no. of processing elements is bounded by degree of concurrency.

$$T_p^{min} = \frac{W + T_o(W, C(W))}{C(W)}$$

* Asymptotic Analysis of parallel programs :-

Example of adding n nos. with n processing elements

$T_p = \theta(\log n)$ $S = \theta(n/\log n)$

* Matrix Computation :-

Matrix - Vector Multiplication :-

```
procedure MAT_VEC(A, x, y)
  begin
    for i := 0 to n-1 do
      y[i] := 0;
      for j := 0 to n-1 do
        y[i] := y[i] + A[i][j] * x[j];
      endfor
  endfor
end MAT_VEC
```

RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI

JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

- Sequential algo needs n^2 multiplications & additions.
- There are 3 ways of doing this parallelly.

i) Row-wise 1D Partitioning :-

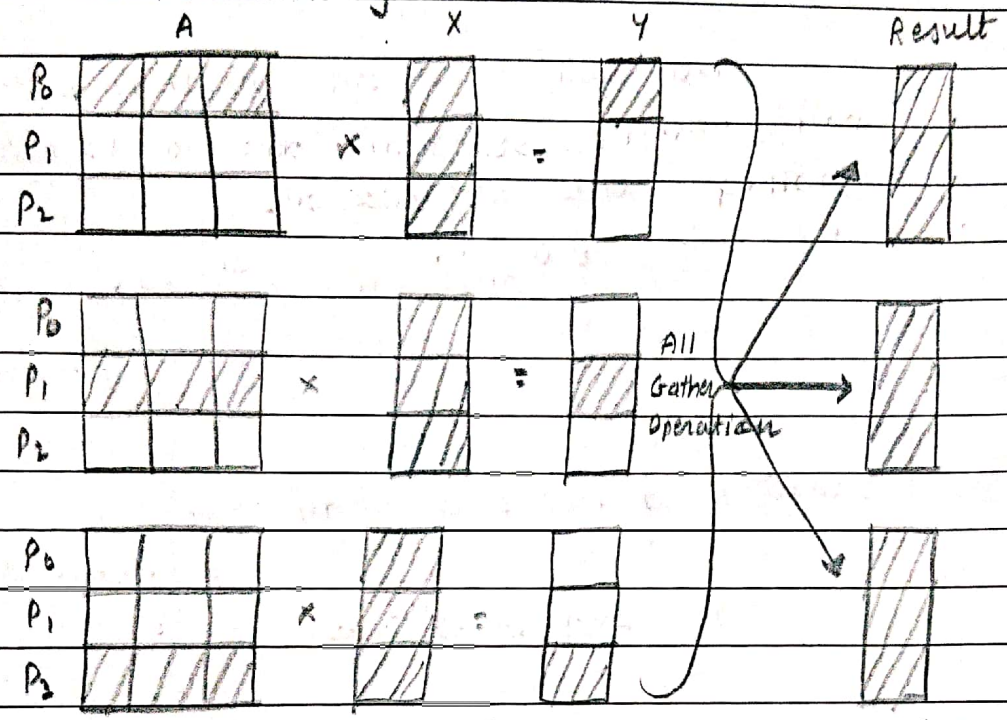
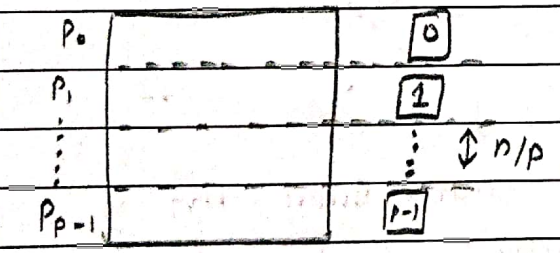
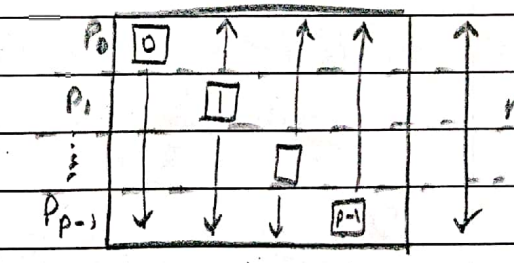


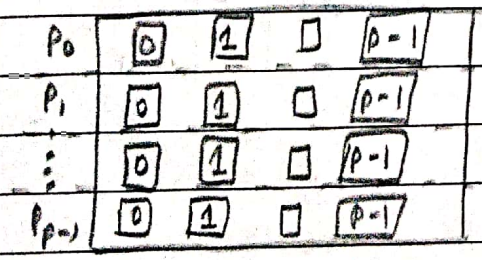
Fig: Computation Scheme for Rowwise Data decomp.



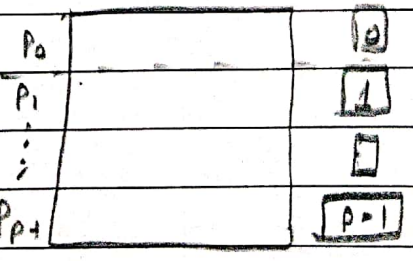
(i) Intial partitation



(ii) All-to-All Broadcast



(iii) Entire vector Distributed

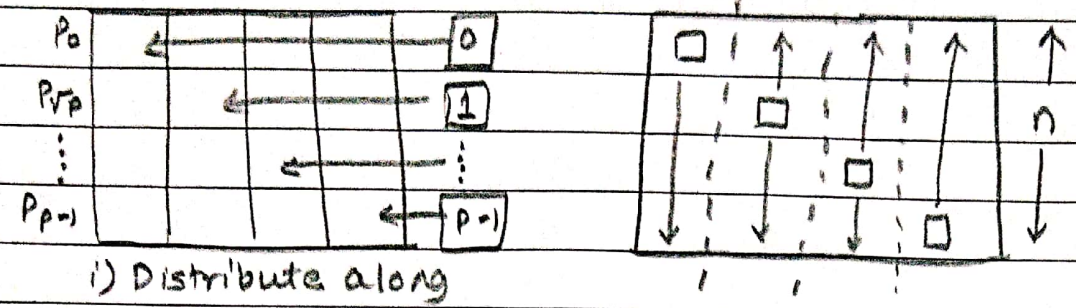


(iv) Final Results.

It is $O(n^2)$ so cost optimized

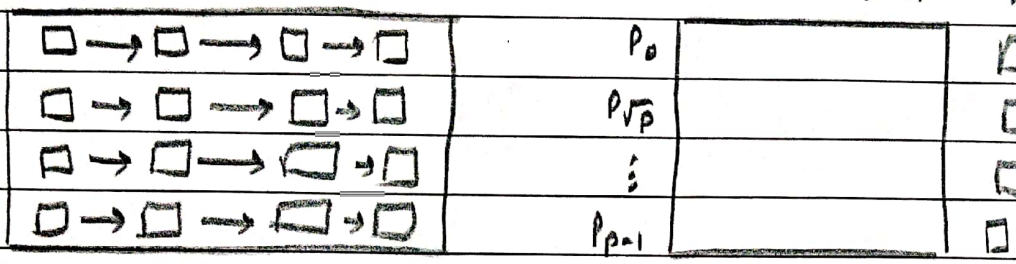
2) Columnwise 1 - D.

3) 2D - Partitioning :-



i) Distribute along diagonal

(ii) One - to - all broadcast

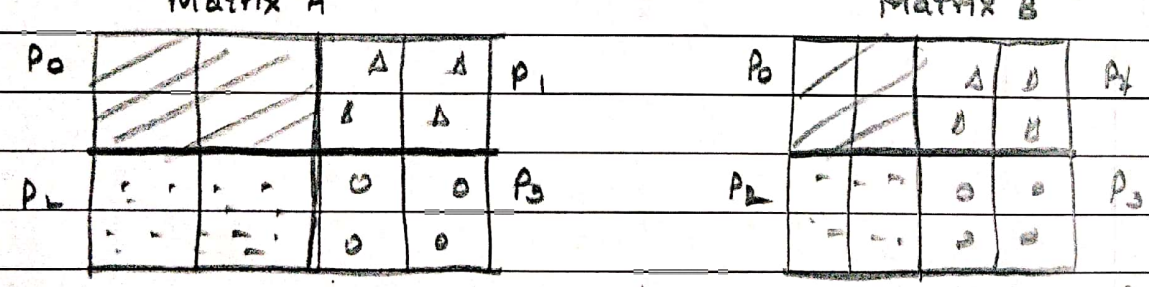


(iii) All - to - one reduction (iv) Final distribution.

Cost is not optimized. It is $O(n^2 \log n)$

* Matrix - Matrix Multiplication :-

- Sequential algo is same with only addition of 3rd loop from 0 to n-1.



Divide in blocks assign to Ps

$C_0 = P_{0,A} \times P_{0,B}$, $C_1 = P_{1,A} \times P_{1,B}$

- We use left shift & Up shift on processes.
- Explain with example.

UNIT 5: CUDA Architecture

* Introduction to GPU :-
- It is the age of parallel computing as many users in today's world do multi-tasking on their phones, tablets, laptops etc.
- To provide novel, fast and rich-experience to the user, developers need to understand & use various parallel platforms.
- CPU performance increasing number of cores/transistors & other features and always has been a reliable way of enhancing performance but not the only way.

• GPUs :-
- Specialized electronic circuit. Helps alter memory rapidly to accelerate creation of images that is given as output to the display device.
- GPUs are in :- Mobile Phones [Adreno/Mali], PCs [AMD/NVIDIA], Consoles [PS4/5, XBOX].
- GPUs are great at image processing & controlling computer graphics.
- They are highly parallel.

• CUDA :-
- Compute Unified Device Architecture.
- Launched by NVIDIA in 2007 as an programming interface providing parallel computation using GPUs.

• CUDA Architecture :-
- Before CUDA, vertex & pixel shaders were used for parallel computing.
- Pixel shader is a GPU component that can operate per pixel.

- Vertex shader is also a GPU component like pixel but it is assembly language specific, used for geometrical operations.
- CUDA was to be used for general-purpose computing, rather than being specifically for graphics.
- 3 Device-level APIs:- i) OpenCL ii) DirectX iii) CUDA Driver API
- Language Level :- 3 :- i) Fortan, ii) C++, iii) C.
- DirectX runs standalone, Rest three need CUDA Driver.
- Then we go software-level i.e CUDA support in OS kernel.
- Then we go hardware-level i.e CUDA parallel compute engine inside NVIDIA GPUs.

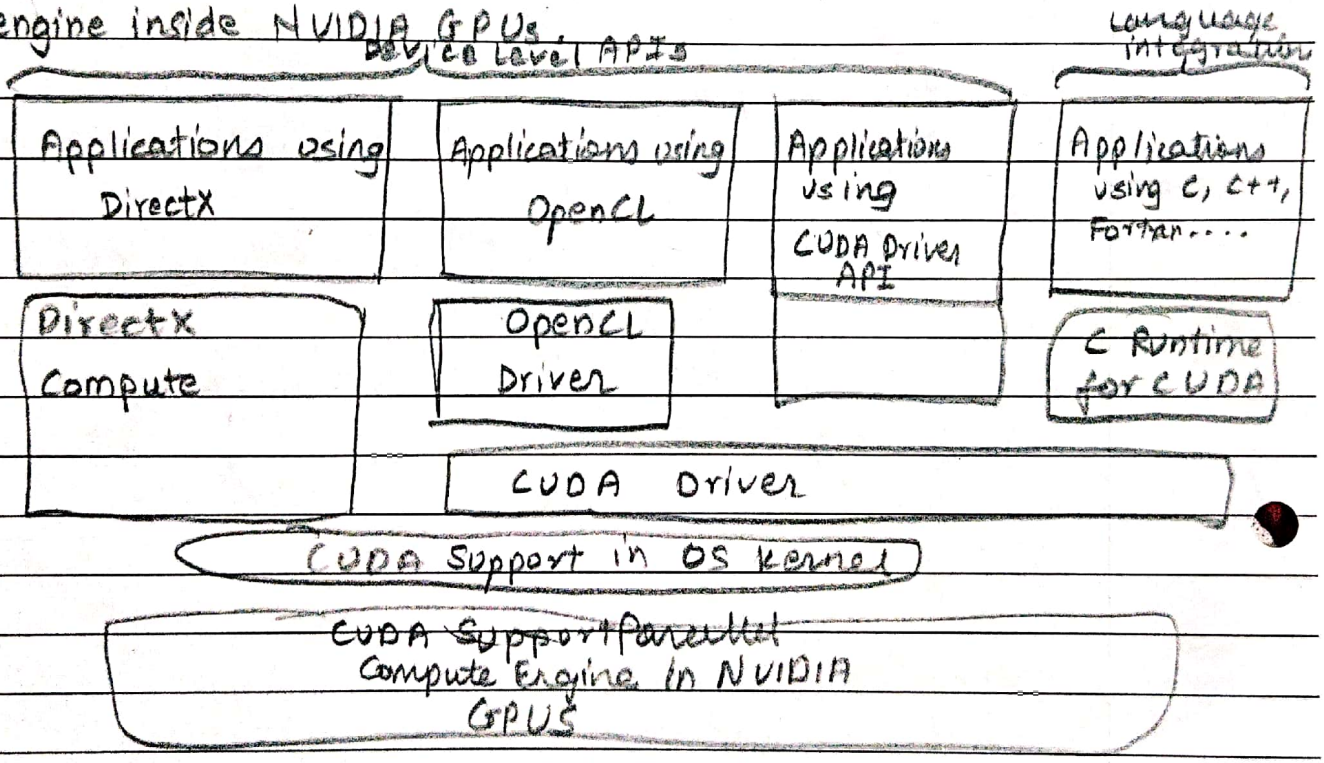
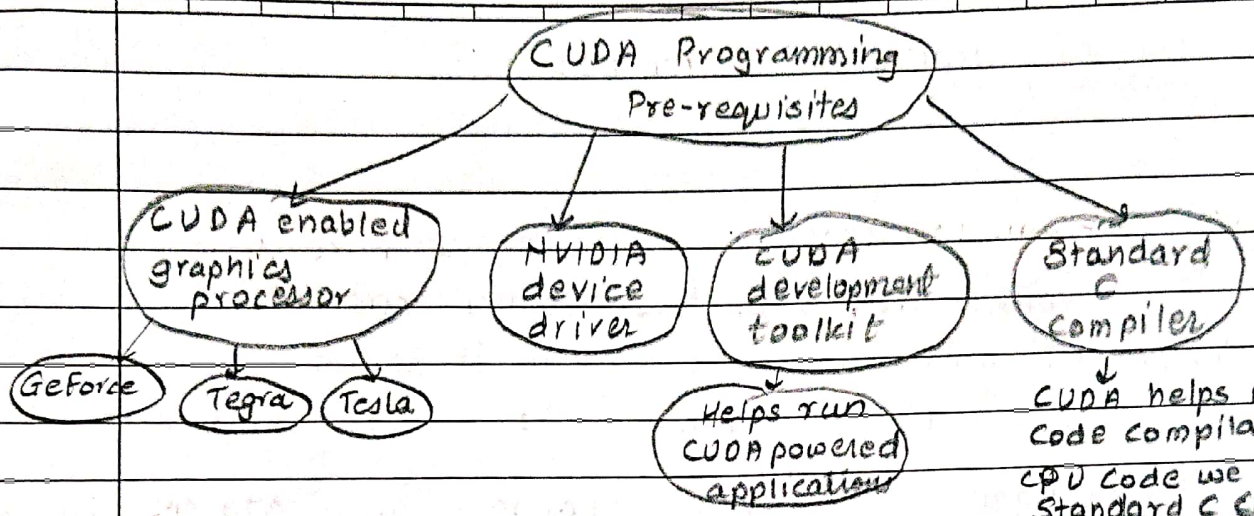


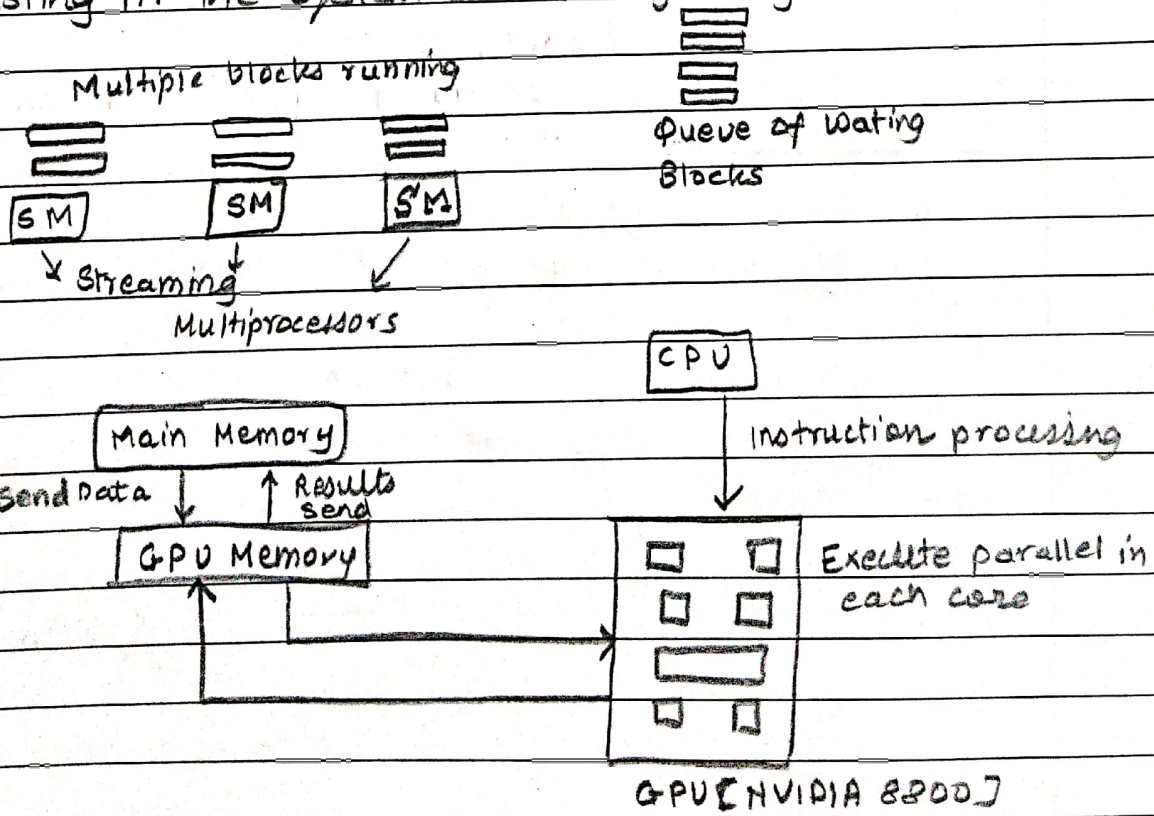
Fig: CUDA Architecture

- Execution units on GPUs are allowed read & write access on a cache known as shared memory.
- CUDA driver consists of parallel computing kernels and functions.
- CUDA C:-

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.



- Working of CUDA C/C++ Program :-
  - Serial code executes in a host [CPU] thread.
  - Parallel code executes in many concurrent device [GPU] threads across multiple parallel processors called as streaming Multiprocessors.
  - Hence to perform parallel execution of code, atleast one CPU is the minimum requirement and one or more GPU existing in the system are arranged by CUDA architecture.



CUDA Workflow

Flow of CUDA C Code :-

1. CPU recognises serial code & parallel code.
2. CPU threads initialized to run serial code & GPU threads for parallel.
3. GPU threads copies data from main mem. to GPU mem for parallel processing.
4. CPU intiates GPU compute kernel.
5. CPU does serial code.
6. GPU's CUDA cores execute kernel in parallel.
7. GPU thread copy data result to main memory.

Recognition → Setup serial & parallel → Copy data M → G → Start GPU kernel by CPU → CPU serial process → GPU parallel process → send data from G → M.

* Example of CUDA kernel program :-

```
#include <iostream>
__global__ void kernel(void) {
}
int main() {
    kernel<<<1, 1>>>();
    printf("Hello world! \n");
    return 0;
}
```

- Global keyword helps call kernel function using host.
- Global helps run the kernel fn on GPU
- We use NVCC compiler to compile this code instead of gcc.
- kernel<<<1, 1>>>() is a CUDA specific syntax that specifies a call to device code.
- Parameters inside "<<< >>>" are called execution configuration

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

- Execution configuration determines no. of threads & blocks.

* CUDA Memory Model :-

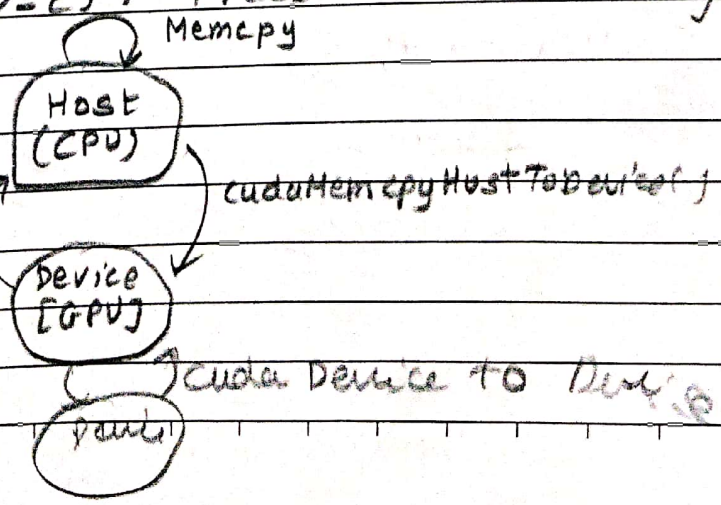
```
#include <iostream>
__global__ void (int* a, int b, int *c) {
        *c = a + b;
}
int main(void) {
    int *c;
    int *dev_c;
    cudaMalloc((void**) dev_c, sizeof(int));
    add<<<1, 1>>>(2, 7, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("2 + 7 = %d \n", c);
    cudaFree(dev_c);
}
```

- The main memory methods used :-

1) cudaMalloc((void**) dev_c, sizeof(int)); :-
- Allocates memory dynamically on device for result.
- We allocate it in terms of bytes using sizeof().

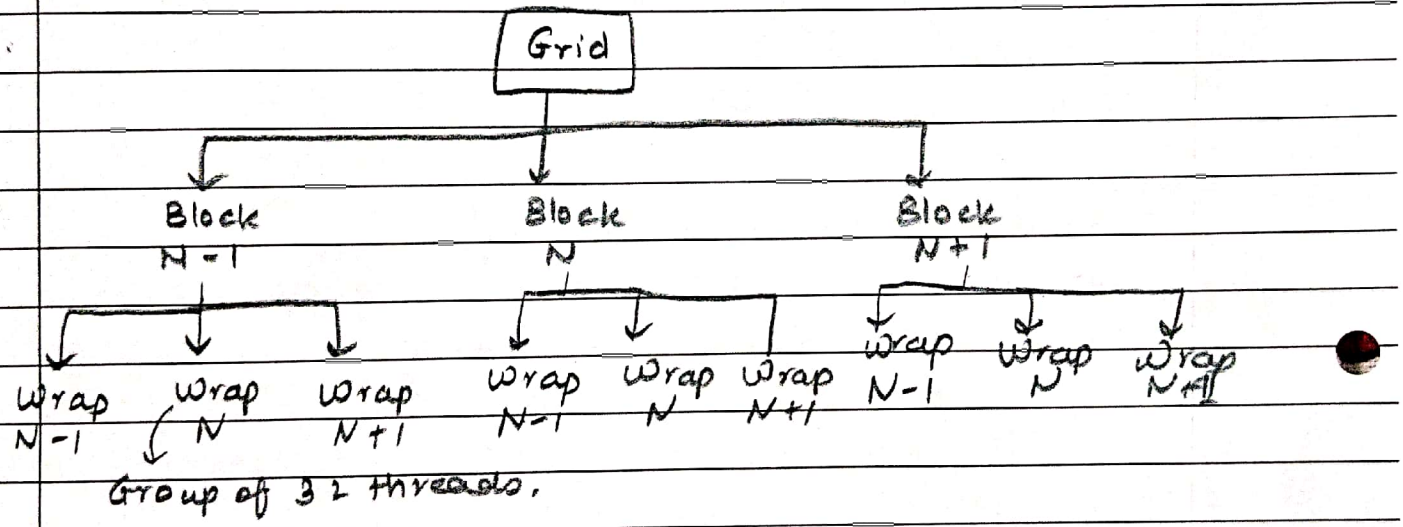
2) cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
- Copy dev_c to c.

3) cudaFree(dev_c) :- Frees allocated memory for dev_c.



- Surround every cuda call with HANDLE_ERROR().
- It is a utility macro that detects errors.
- It helps in debugging.

* Manage Communication & Synchronization:-
- CUDA is ideal when little to no inter-thread/block communication is ~~thread~~ needed.
- For inter-thread on chip resources with explicit primitives are used.
- For inter-block multiple kernels are invoked, using off-chip global memory.
- Methods used for Synchronization : cudaDeviceSynchronize()
  cudaDeviceReset()



- Simultaneous Multiprocessors switch between wraps.
- Blocks run in any order, only a subset of block executes anytime.
- Blocks run on SMs, and are allocated in a round-robin format.
- Wraps exhibit SIMD.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

• Parallel Programming in CUDA C [Theory]:-

I] Threads :-

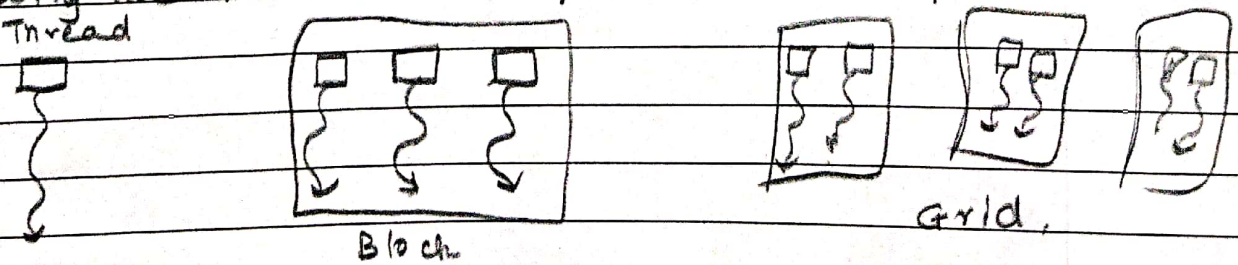
- Fundamental unit of a program.
- Thread is a lightweight process, handles 1 task at a time.
- Helps bring concurrency in application.
- Give example of vector addition using threads.
- Thread info stored in threadIdx i.e a built-in CUDA variable.
- ThreadIdx is accessed as :- int tid = threadIdx.x
- Thread id is unique within each block.
- Each thread that executes kernel can be referenced by its ID.

II] Blocks :-

- Threads are organized into blocks, blocks are refered as blockIdx in CUDA runtime.
- Threads within block synchronize, on a modern GPU a block may contain 1024 threads.
- Using block ID & thread ID we can make unique thread ID for each kernel.
- So kernel <<< Number_of_blocks, Number_of_threads >>> () is how we call a kernel.
- ∴ kernel <<< 256, 1 >>> () gives 256 blocks.
- Specifically <<< Number of blocks per grid, Number of threads per block >>> ().

III) Grid :-

- Grid allows to define a block in two dimensions.
- Mostly used in matrix multiplication / other operations.



UNIT 6 : HIGH PERFORMANCE APPLICATIONS

* Parallel Sorting :-
- Sequential algorithm :-

```
procedure BUBBLE_SORT(n)
begin                                    TC :- O(n^2)
    for i := 0 to n-1 do
        for j := i+1 to n-i-1 do
            compare-exchange (aj-1, aj);
        endfor
    endfor
end BUBBLE-SORT(n)
```

- Odd-Even Transposition :-
- This algorithm sorts an array of n elements in n phases (if n is even), each phase requiring n/2 comparisions.
- It alters between odd & even phase, in odd phase, only elements at odd indexes are compared, and in even; even.
- procedure ODD-EVEN(n):

```
begin
    for i := 0 to n-1 do
    begin if i is odd then
            for j := 0 to n/2 - 1 do
                compare-exchange (aj, aj+2)
        if i is even then
            for j := 1 to n/2 - 1 do
                compare-exchange (aj, aj+2)
    endfor
end ODD-EVEN
```

[only write end for statements that have begin]

- The algorithm can be made parallel.

```
procedure ODD_EVEN_PAR(n)
begin
    for id := process_label
    for i := 1 to n do
    begin
        if i is odd then
            if id is odd then
                compare-exchange_min(id+1);
            else
                compare_exchange_max(id-1);
        if i is even then
            if id is even then
                compare-exchange_min(id+1);
            else
                compare-exchange_max(id-1);
    endfor
end ODD_EVEN_PAR
```

- Here p=n and complexity is $O(n^2)$ which is not cost optimal.
- If p = log n then optimal cost performance is achieved.

Example

5 4 3 2 Eve
4 5 2 3 odd
4 2 5 3 Even
2 4 3 5 Sorted?

* Distributed computing:-

• Document classification:-

- Process of assigning predefined labels/categories to documents which include unstructured & semi-structured.

• Procedure:-

1) Set of pre-classified documents used as training set.
2) A classification scheme is derived.
3) Scheme is refined using testing process.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

4) New docs are now classified,
- Doc classification on sequential CPUs takes lot of time.
- GPU can be used as parallel unit for this task.
- Parallel kNN in CUDA/MPI can be used here.

* High Performance Frameworks :-
- Kubernetes :-
- An open source system for handling development lifecycle.
- Helps in automating deployment, scaling and management of containerize application that also support distributed computing.
- Helps in not compromising security or reliability.
- Saves time on infrastructure management.
- Acts as an orchestrator for containers.

- Parallel computing for AI/ML - Kubeflow :
- ML Toolkit for Kubernetes.
- Keeps workflows simple, portable & scalable.
- Supports Jupyter notebooks & tensorflow model convinient.
- Adds portability in ML Deployments.

* Scope of parallel Computing :-
1) ~~Engee~~ Engineering & Design :- Complex Math solving, Optimization,
2) Scientific Application :- DNA structures, Astrophysics, Bioinformatics.
3) Commercial :- Stock, Data Mining, Large scale servers.
4) Computer system :- Intrusion detection, Modern automobile, P2P Networks.

* Parallel Merge Sort :

- Divide array into two parts. 1st part = array elements with odd index & 2nd = array elements with even index.
- Consider another array B.
- We define merge(A, B) as:

  merge(A, B) = { a1, b1, a2, b2, ..., an, bn }

  If A = { 1, 2, 3, 4 }; B = { 5, 6, 7, 8 } then

  Merge (A, B) = { 1, 5, 2, 6, 3, 7, 4, 8 }
- Now, Join(A, B) = (Merge(A, B), Odd-Even (A, B))
- Algorithm:

```
>>> procedure ODD-EVEN(A, B, S)
    begin
    if A and B are of length 1 then
        Merge A and B using compare-exchange operation
    else
      begin
      compute Sodd and Seven in parallel do
      Sodd = Merge (Aodd, Bodd)
      Seven = Merge (Aeven, Beven)
      Seven-odd = Join (Sodd, Seven)
      end
    end ODD-EVEN
```

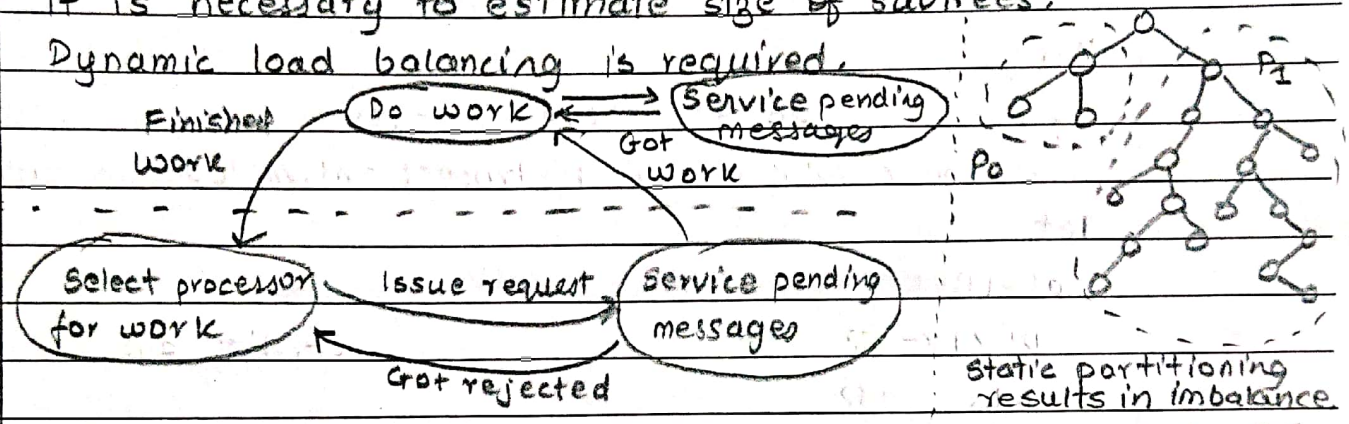
- Example:-
- Let S = { 2, 3, 6, 10, 15, 4, 5, 8 }; A = { 2, 6, 15, 4 }
  B = { 3, 10, 5, 8 };
  Aodd = { 2, 15 }; Aeven = { 6, 4 }; Bodd = { 3, 5 }; Beven = { 10, 8 }
- Now, Merge (Aodd, Bodd) = { 2, 3, 5, 15 }
  Merge (Aeven, Beven) = { 6, 10, 4, 8 }
  ∴ Merge (A, B) = { 2, 6, 3, 10, 5, 4, 15, 8 }
  Now, Join (A, B) = [Merge (A, B), Odd-Even (A, B)]

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

We already know Merge (A, B), ~~Join~~ Odd-Even will sort (A, B) and give the final ans :-
Odd-even (A, B) = { 2, 3, 4, 5, 6, 8, 10, 15 }

- Parallel merge-sort takes $O(\log n)$ time on using n processor, hence total time : $n * O(\log n) = O(n \log n)$
- It is cost-optimal.

* Parallel DFS :-
- Idea :- Different sub-tree can be searched concurrently.
- It is necessary to estimate size of subtrees.
- Dynamic load balancing is required.



- Work is split by splitting stack in two.
- Neither of split stack should be small, but both equal.
- Node splitting :- Select nodes from bottom of stack.
- Stack splitting :- Select nodes from each level of stack.
- Stack splitting is favored.
- Several ways to do load balancing :-

1) Asynchronous round robin :- Counter for each processor, requests made in round-robin style. $V(p) = O(p^2)$ [Poor]
2) Global round robin :- Global couter + round robin. $V(p) = O(p)$ [Poor performance due to limited resources; Contention]
3) Random polling :- Randomly select processor. $V(p)$ is not bounded. [Desirable Performance]

- Po has all work, and has a weight associated with it.
- It will partion its work and give it to another processor.
- In doing so it loses half of its weight, and weight is gained by recipent processor.
- After completion of work, Po will return to its normal weight & termination is signalled.
- Work W is split into ψw & $(1-\psi)w$ and an constant α $(0 < \alpha \leq 0.5)$ is used such that $\psi w \geq \alpha w$ & $(1-\psi)w > \alpha w$.
- α sets lower bound for load imbalance.

* Parallel BFS :-

- Key ideas :- 1) Do BFS Level-By-Level not vertex by vertex. [Level Synchronization]. 2) Process entire level in parallel.
- Let us see the pseudocode :-

```
ParallelBFS(G ∈ (V, E), s ∈ V)
    D[v] ← ∞
    D[s] ← 0
    l ← 0   // Level Counter
    F0 ← {s} // Add source to frontier
    while Fl ≠ ϕ do
        Fl+1 ← ϕ // Next level frontier
        processlevel(G, Fl, Fl+1, D)
        l ← l+1 // Go to next level
    return D
```

```
processlevel(G, Fl, Fl+1, D)
    if |Fl| > ϕo then // If bag is big then split
        (A, B) ← bagSplit(Fl)
        spawn processlevel(G, A, Fl+1, D)
        processLevel(G, B, Fl+1, D)
        sync
    else // If bag ain't big
        for v ∈ Fl do
            par-for (v, w) ∈ E do
                if D[w] = ∞ then
                    D[w] ← l+1
                    bagInsert(Fl+1, w)
```

- Frontier holds all the nodes of a single level.
- while $F_l \neq \phi$: the span is not more than diameter of graph.
- We use divide-and-conquer while splitting the bag that holds frontier.
- A bag is a multi-set data structure.
- Cost: $W = O(|V| + |E|)$ Total cost = $O(d \times \log |V \times E|)$ (d ↓ diameter)

source: Udacity