Name - Ashutosh soni

Id - 2018UCP1505.

Que ⇒

(a)

Ans ⇒

Parallel prefix problem is about finding sum of consective element of an array coming earlier to that index.

Let suppose,

we are given n values stored in array as

a1, a2, a3 - - - - - - an.

the prefix sum problem is to compute. n quantities.

a1, a1+a2, a1+a2+a3, - - - - , a1+a2+ - - an.

- Parallel solution for this problem.

→ first we divide our initial array to subarrays equal to no. of threads. Each thread then linearly calculate the prefix sum. for the assigned sub-array. This prefix sum will be less than the actual sums because elements before the start of a particular sub-array are ignored. The last element then stored in another array. Again the new array is formed Now we calculate the prefix sum for this newly created array. simply as we calculate prefix sum for earlier subarrays.

Example Given :-

5 1 0 2.

Pass - 1

1) $\boxed{5}$ $\boxed{1}$ $\boxed{0}$ $\boxed{2}$

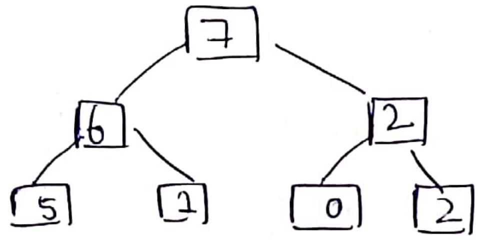
2) 5 $\boxed{6}$ ~~0~~0 $\boxed{2}$

3) 5 6 0 $\boxed{8}$

Name - Ashutosh Soni

Id - 2018UCP1505.

Pass 2. Down Tree for earlier.

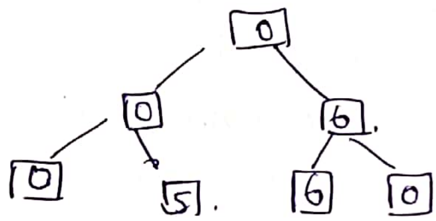


clear from that

4) 5. 6 0 $\boxed{0}$ } clear

5). 5 $\boxed{0}$ 0 $\boxed{6}$ }

6) 0 $\boxed{5}$ $\boxed{6}$ $\boxed{6}$



(1b)

<u>Bubble sort</u>

procedure BUBBLE_SORT(n)

begin

id: = process's level.

for i:=1 to n do.

if i is odd and id is odd then.

compare-exchange_min (id+1);

else.

compare-exchange_max (id-1);

Name - Ashutosh Soni
Id - 2018UCP1505

if i is even and id is even then.
compare-exchange-min (id+1);
else
compare-exchang-max (id-1);

end for.

end BUBBLE_SORT.

above is the algorithm of Bubble sort.
we are dividing it into two phases even-phase and odd phase. Even phase or even numbered processes exchange number with the number adjacent to the right and same with odd phase or odd numbered phase.

ex:

9 7 3 8 5.
7 9 3 8 5
7 3 9 5 8
3 7 5 9 8
3 5 7 9 8 ~ finalized ascending array

Performance:

Parallel Bubble sort sort the array in ~~asse~~ ascending or descending order in O(n). whereas sequential Bubble sort sort the array in $O(n^2)$. Hence from looking at order of both we can clearly see the parallel bubble sort is far better then sequential Quick sort.

Name- Ashutosh Soni
Id-2018UCP1505

Que2:
(a)
Ans: General Semaphore:-

A semaphore whose integer component can take arbitrary non-negative values is called a general semaphore.

Mathematically,
for Semaphore.

$S.V = K$ where $K >= 0$

& $S.L = \phi$

is known as general semaphore.

(b)
Readers-Writers Problem using Monitors..

Algorithm for Readers-Writers problem using Monitors:-

```
monitor Reader_Writer
    integer readers ← 0
    integer writers ← 0.
    condition OKtoRead, OKtoWrite

    operation StartRead.
        if writers ≠ 0. or not empty (OKtoWrite).
            wait((OKtoRead).
        readers ← readers+1.
        signal (OKtoRead)

    operation. EndRead.
        readers ← readers+1.
        if readers = 0.
            signal(OKtoWrite).
```

Name - Ashutosh Soni
Id - 2018UCP1505

operation. StartWrite.
    if writers ≠ 0 or readers ≠ 0
        wait (OK to Write).
    writers ← writers+1

operation EndWrite.
    writers ← writers-1.
    if empty (OKtoRead).
        then signalC (OKtowrite).
        else signalC (OKtoRead).

reader
p1: RW. StartRead.
p2: read the data.
p3: RW. endRead.

writer.
q1: RW. StartWrite.
q2: write the data.
q3: RW. endWrite.

above is the algorithm for readers writers problem. where we declare two condition variable

1) OKtoRead
2) OKtoWrite

and 2 integer value which will contain Readers numbers & writers number. & there are four operations

1) StartRead
2) endRead
3) StartWrite
4) endWrite.

- Readers are processes which require to exclude writers only. NO condition with Readers. whereas writers are the process which require to exclude both readers and the writers. Accordingly we we wait and signal to get desired result mentioned above.

Scanned by CamScanner

Name - Ashutosh soni
Id - 2018UCP1505

Que 3:
Ans ⇒
(a)

In OPENMP,

Data scope Attribute clauses are used to define the scope of variable. By using data-scope Attribute we limit the scope of variable like is visible outside or not like that.

These are many Data scope Attribute clases listed below.

1) PRIVATE. 2) FIRSTPRIVATE. 3) LASTPRIVATE.
4) SHARED 5) DEFAULT 6) REDUCTION.
7) COPYIN.

Data-Attribute clauses in OPENMP is used in conjuction in several directives to control the scoping of the variable. These constructs provide the ability to control the data environment during execution of parallel constructs.

They are only effective within lexical / ~~ext~~ static extent.

- PRIVATE clause :- In this clause the variable declared is private to the thread.

  private (list) ;

- SHARED clause :- In this clause the variable declared is shared to all the threads in the team.

  Shared (list).

- DEFAULT clause :- This clause allows the user to specify a default scope for all variables in the lexical extent of any parallel region.

  ~~def~~ default (shared | none).

Name-Ashutosh soni
Id-2018UCP1505.

- FIRSTPRIVATE: This clause combines the behavior of the PRIVATE clause with automatic initialization of the variables in its list.

firstprivate (list)

- LASTPRIVATE: This clause combines the behaviour of the PRIVATE clause with a copy from the last loop iteration or selection to the original variable object

lastprivate (list).

- REDUCTION → This clause perform as reduction on the variable that appears in its list. A private copy of each variable created for each thread.

reduction (operator : list).

- COPYIN: specifies that the data in the master thread of the team to be copied to the thread private copies of the common block at the beginning of the parallel region

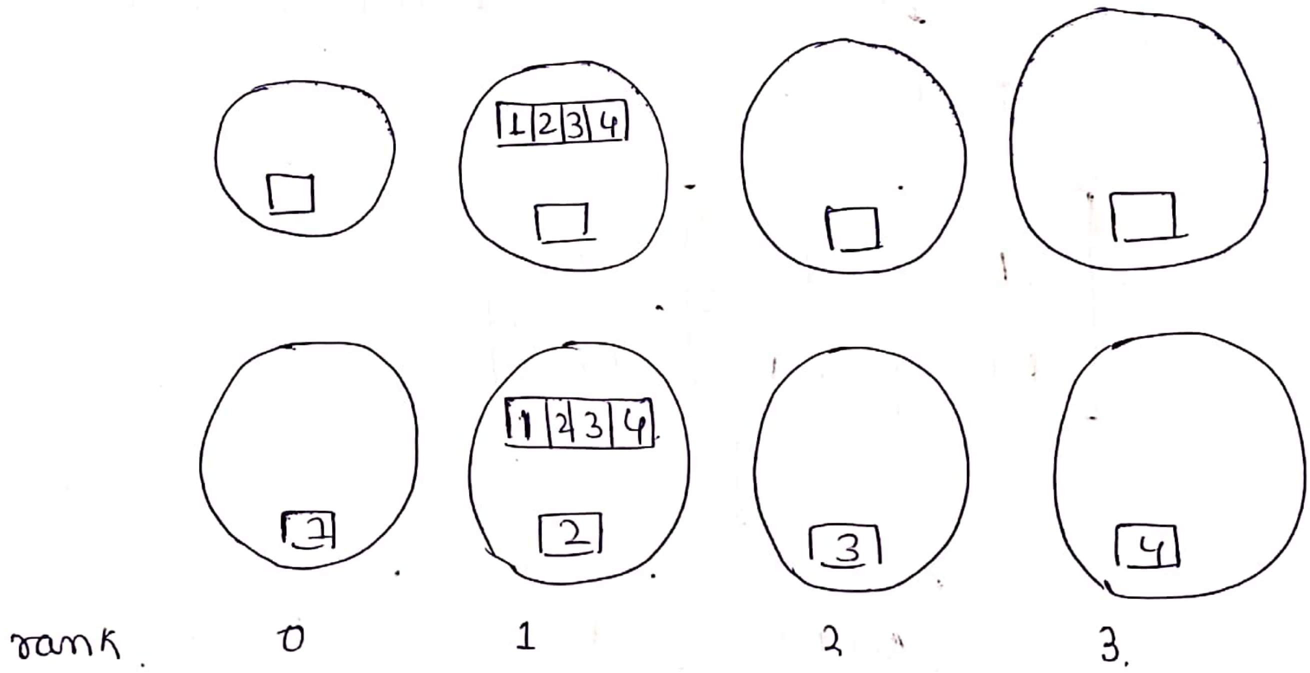
COPYIN (list).

Name - Ashutosh soni
Id - 2018 UCP1505

(b)

(i) Scatter:
It is a type of one to all communication: In this different data sent to each process in the communicator in (rank order):

Syntax:
int MPI_Scatter (void* sendbuf, int send count, MPI_DATATYPE sendtype, void* recvbuf, int recvcount, MPIDatatype recvtype, int root, MPI_Comm comm).



this is the example of Scatter clause.

Name - Ashutosh Soni
ID - 2018UCP1505

(ii) Scan → Parallel Prefix.
Use the value of result from previous process.
in the next process.

MPI_Scan.

A_0	B_0	C_0
A_1	B_1	C_1

⟹

A_0	B_0	C_0.
A_0+A_1	B_0+B_1	C_0+C_1.

(iii) All to All Reduction.
→ In this there is no Root process & all
get Result.

A_0	B_0	C_0
A_1	B_1	C_1

⇓.

A_0+A_1	B_0+B_1	C_0+C_1
A_0+A_1	B_0+B_0	C_0+C_1.

Scanned by CamScanner

Name-Ashutosh soni
Id-2018UCP1505

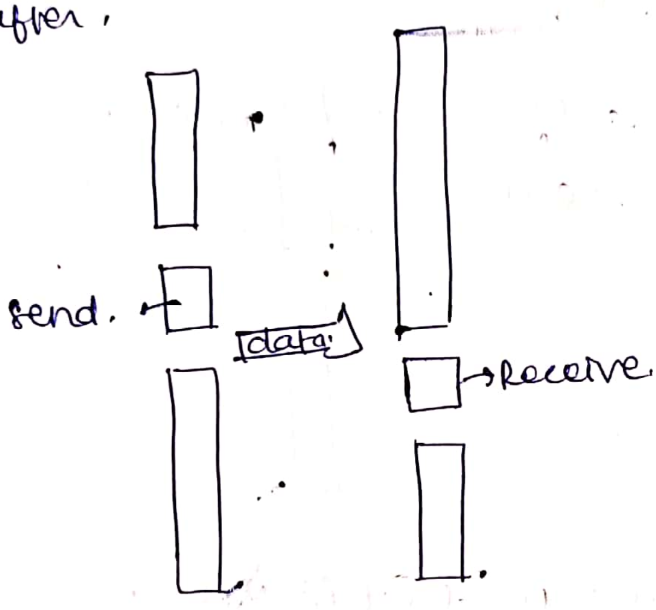
(c).

Ans→.

Send & Receive in MPI for both blocking & non-blocking operation.

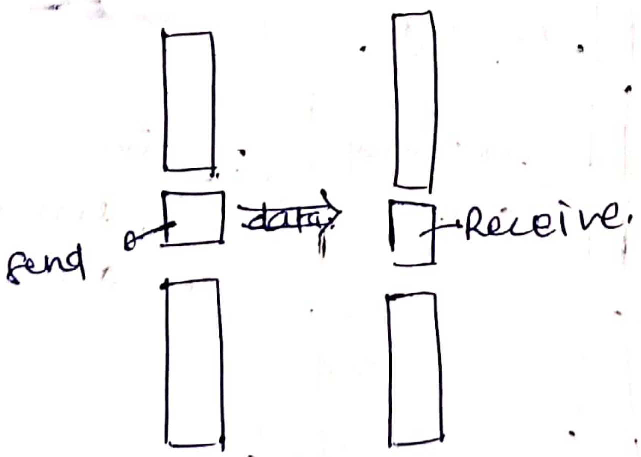
1) for Blocking & Buffered operation.

• In this sending process returned after data has been copied into communication buffer.



2) for ~~Non~~ buffered Blocking operation

sending process blocks until. ~~blocks~~ matching receive operation has been encountered.

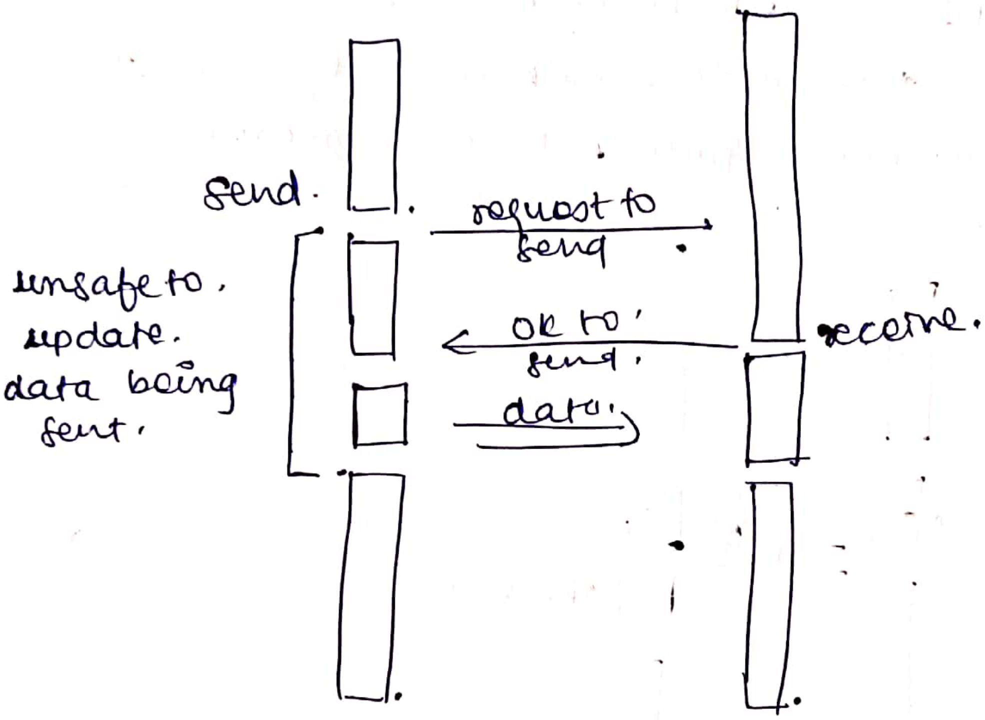


before that process has blocked until rec. for that send get encountered.

Name - Ashutosh soni
Id - 2018UCP1505

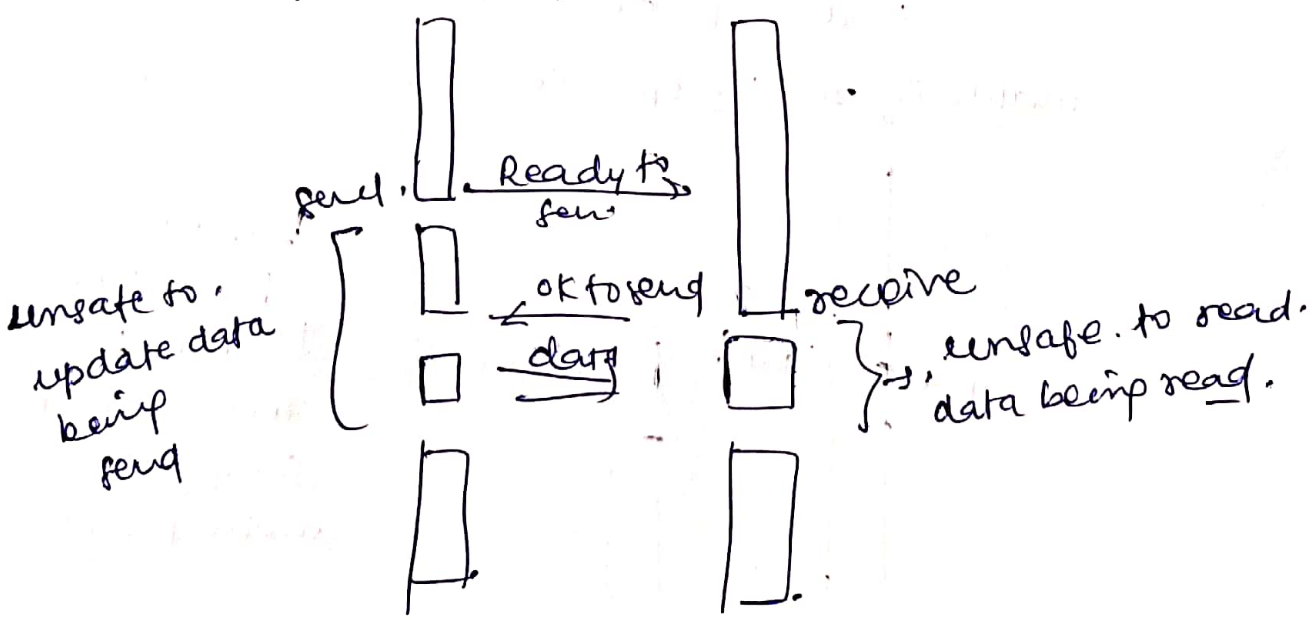
(c). Buffered Non-Blocking opeation

→ Sending process returns after initializing DMA transfer to buffer. This operation may not be completed on return.



(d). Non-Buffered Non-Blocking operation.

• In this there is no Buffer stored for sending & receiving.



Name - Ashutosh soni
Id - 2018UCP1505

In ~~Blocking~~ Buffered case there is buffer to store data. whereas not in case of Non-Buffered operation case.

- Send & Receive semantics assured by corresponding operation in Blocking operation
- Programmer must explicitily ensure semantics by polling to verify ~~oper~~ completion in case of Non-Blocking operation.

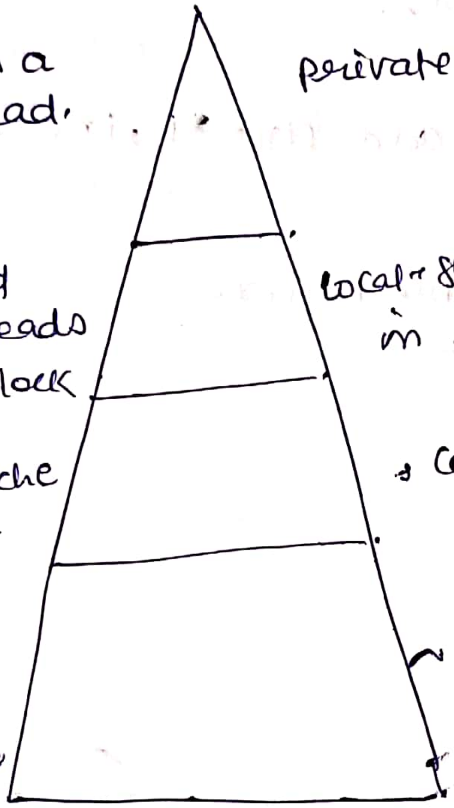
Name-Ashutosh Soni

Id -2018UCP1505

Ques 4:

(a)

Ans: Memory Hierarchy terminology in CUDA and OpenCL.



CUDA	OPENCL.
local → within a thread.	private → within a work-item.
shared → shared between threads in a thread block	local → shared between work-item in a work group.
constant → a cache for constant memory	constant → a cache for constant memory.
device → shared between all thread blocks	Global → shared between all work groups.

Name - Ashutosh Soni

Id - 2018UCP1505

(b)

Ans.

Program for addition of two square matrix in CUDA.

```
# include <stdio.h>
# include <cuda.h>.

__global__ void MatAdd (int A[][N], int B[][N],
                        int C[][N]) {

    int i = threadId.x;
    int j = threadId.y;

    C[i][j] = A[i][j] + B[i][j];

}

int main() {
    int A[N][N] = {{1, 2}, {4, 5}};
    int B[N][N] = {{4, 5}, {5, 6}};
    int C[N][N] = {{0, 0}, {0, 0}};

    int *(pA)[N], *(pB)[N], *(pC)[N];

    cudamalloc (pA, A, (N*N)*sizeof(int),
            cudamemcpyHostToDevice);

    cudamal
```

Name - Ashutosh Soni
Id - 2018UCP1505

```
cudamalloc ((void**)&pA, (N*N)*(sizeof(int)));
CudaMalloc ((void**)&pB, (N*N)*(sizeof(int)));
CudaMalloc ((void**)&pC, (N*N)*(sizeof(int)));

cudamemcpy ( pA, A, (N*N)*sizeof(int), cudaMemcpyHostTo
        Device);.

cudamemcpy (pB, B, (N*N)*sizeof(int), cudamemcpyHostTo
        Device);

cudamemcpy ( pC, C, (N*N)*sizeof(int), cudaMemcpyHost
        ToDevice);

int numBlocks=1;
dim3 threadsperBlock(N,N);

MatAdd <<<numBlocks, threadsperBlock>>>
                (A B C);.

cudamemcpy ( C, pC, (N*N)*sizeof(int),
                cudamemcpyDeviceToHost);

int i, j;
    printf(" (");.
```

Name - Ashutosh Soni
Id - 2018UCP1505

```
for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
                printf("%d ", c[i][j]);

        }
        printf("\n");
},

cudafree (pA);
Cuda free (pB);
cuda free(pC);

printf("\n");

return 0;
}
```

Performance on shared memory :-

As in shared Memory we.

- Partition data into subsets that fit in shared memory.
- Handle each data subset with one thread block by:
  - Loading the subset from global memory to shared memory

Name - Ashutosh soni
Id - 2018UCP1505.

- Performing the computation on the subset from. shared memory, each thread can efficiently multipass overmany data Element
- Copying results from shared memory to global memory.

by doing this,
we can increase the perfore the systemalot.