Ex. No: 1	OPENMP – BASIC PROGRAMS SUCH AS 1. VECTOR ADDITION, 2. DOT PRODUCT.
Date: 2 Feb	

AIM: To gain practical experience in parallel programming in OpenMP, understand the principle of vector addition and dot product computation in parallel environ.

ALGORITHM:

Vector Addition:

1. Initiate array 'A' & 'B' of size 'N'.
2. Create another array 'C' of size 'N'.
3. Use OMP directives to distribute the addition operation.
4. Return the result array.

Dot Product:

1. Initiate 2 arrays 'A' & 'B' of size 'N'.
2. Use OMP directive to distribute the multiplication operation, with thread for parallelism.
3. Combine partial dot products & ~~vector~~ return result array

Ex. No: 2	OPENMP – 1. LOOP WORK-SHARING AND 2. WORK-SHARING.
Date: 9 Feb	

AIM: To parallelize loop iterations & sections of code using OMP directives to efficient utilization.

ALGORITHM:

Loop Work sharing :

Step-1 : Identify computationally intensive loop

Step-2 : Use OMP directives to distribute loop iterations.

Step-3 : Each thread executes a subset of loop iteration.

Section Work sharing :

Step-1 : Identify distinct section of the code that can be parallelized.

Step-2 : Assign each section to a thread.

Step-3 : Each thread executes a section parallely.

Ex. No: 3	OPENMP – 1. COMBINED PARALLEL LOOP REDUCTION AND 2. ORPHANED PARALLEL LOOP REDU'
Date: 16 Feb 24	

AIM: To implement combined parallel loop reduction and orphaned parallel loop reduction techniques.

ALGORITHM:

Step-1 : Identify a loop with a reduction operation on a shared variable

Step-2 : Parallelize loop with one directive such as # pragma OMP parallel for reduction.

Step-3 : Distribute tasks among threads in chunks.

Step-4 : Use reduction clause to accumulate result

⟹ for orphaned parallel loop :-

For Orphaned, Step 2 & 2 are same

Step-3 : Each thread calculates local sum of prod.

Step-4 : Else, a initial section to safely accumulate local sums the final result to avoid conditions.

Step-5 : Output the computed result.

Ex. No: 4	OPENMP – MATRIX MULTIPLY (SPECIFY RUN OF A GPU CARD, LARGE SCALE DATA ... COMPLEXITY OF THE PROBLEM NEED TO BE SPECIFIED).
Date: 1 Mar	

AIM: To implement matrix multiplication using OMP and execute it on GPU.

ALGORITHM:

17/45

S1. Initialize 'N' x 'N' matrices 'A' & 'B' random values

S2: Allocate memory for result matrix 'C'.

S3: Use OMP directives to parallize matrix multiplication.

S4: Distribute the computation of each element

S5: Ensure proper synchronization to avoid hazards.

S6: Offload computation to GPU using OMP directives.

Ex. No: 5	MPI – BASICS OF MPI.
Date: 15 Mar	

AIM: To understand basics of message passing interface (MPI) programming paradigm.

21/45

ALGORITHM:

S1 : Initialize MPI environment

S2 : Identify size & rank of the world.

S3 : Demonstrate point-to-point communication

S4 : Finalize MPI environment.

Ex. No: 6	
Date: 22 Mar	**MPI – COMMUNICATION BETWEEN MPI PROCESS.**

AIM: To establish communication between MPI process for exchanging data.

ALGORITHM:

S1 : Initialize MPI environment.

S2 : Determine the rank of each process.

S3 : Depending on process rank.

⇒ If it is root process & world rank == 0) initializes & broadcast data using the function.

⇒ If not, recieve the broadcasted data using function.

S4 : Print message to indicate broadcasting

S5 : Finalize MPI environment.

PROGRAM:

```
#include <mpi.h>
```

Ex. No: 7	MPI – COLLECTIVE OPERATION WITH "SYNCHRONIZATION".
Date: 29 Mar	

AIM: To utilize collective operations in MPI with synchronization mechanism for efficiency.

ALGORITHM:

S1 : Initialize MPI environment

S2 : Use collective operations such as 'MPI - Bcast', 'MPI - Reduce', 'MPI - Scatter', with synchronization.

S3 : Co-ordinate data exchange & computation among MPI process.

S4 : After the barrier, print message indicate that process knows all MPI processes have reached on barrier.

S5 : Finalize MPI environment.

PROGRAM:

Ex. No: 8	MPI – COLLECTIVE OPERATION WITH "DATA MOVEMENT".
Date: 12 Apr	

AIM: To utilize collective operations in MPI for efficient data movement among MPI procs.

ALGORITHM:

S1 : Use collective operations such as 'MPI - Scatter', 'MPI.- Alltoall'.

S2 : Distribute or gather data among MPI procs.

S3 : If current proc is ~~root~~ root proc, use MPI- gather to gather values from all procs into buffer.

S4 : If not root, use MPI- gather to send my proc's value to root proc.

S5 : Print gathered values.

S6 : Finalize MPI proc environment

Ex. No: 9	MPI – COLLECTIVE OPERATION WITH "COLLECTIVE COMPUTATION".
Date: 19 Apr	

AIM: To demonstrate collective operations in MPI for performing collective computation

ALGORITHM:

S1 : Initialize MPI process.

S2 : Use operations like 'MPI-reduce' & 'MPI-gather'

S3 : Perform computation across all MPI process.

S4 : Each process sends its rank to the reduction operation with root process using MPI - reduce.

S5 : If current process is root, print result of the reduction process, operation which represents sum of all ranks across MPI process.

S6 : Finalize MPI Environment.

Ex. No: 10	MPI – NON-BLOCKING OPERATION.
Date: 26 Apr	

AIM:

To implement Non-blocking communication on MPI for overlapping computation.

ALGORITHM:

S1: Initialize MPI environment.

S2: Use non-blocke communication function, like 'MPI-Isend' & 'MPI-Irecv' for async communication

S3: Ensure computation while, communication is progress.

S4: If current procss is root (Master), initialize buffer value & send to slave procss using MPI-Isend

S5: If current procss is 'slave', declare variable to recieved value storage. Initialize MPI-Irecv to async recieve msg from 'Master' procss.

S6: Print appropriate msg to indicate procss completion.

S7: Finalize MPI environment.