```
Results for N=3;

#Check A row = 0, #Nodes = 3
     04     03     02
#Check C row = 0, #Nodes = 3
     75     68     43
#Check A row = 1, #Nodes = 3
     10     08     06
#Check C row = 1, #Nodes = 3
    204    184    116
#Check A row = 2, #Nodes = 3
     18     15     12
#Check C row = 2, #Nodes = 3
    387    348    219
Brodcasted matrix B:     06     06     04
     09     08     05
     12     10     06
Gathered matrix D:     75     68     43
    204    184    116
    387    348    219
Run time on  3 nodes: 0.000051



Results for N=5;

#Check A row = 0, #Nodes = 5
     06     05     04     03     02
#Check C row = 0, #Nodes = 5
    350    360    330    260    150
#Check A row = 1, #Nodes = 5
     14     12     10     08     06
#Check C row = 1, #Nodes = 5
    900    920    840    660    380
#Check A row = 2, #Nodes = 5
     24     21     18     15     12
#Check C row = 2, #Nodes = 5
   1650   1680   1530   1200    690
#Check A row = 3, #Nodes = 5
     36     32     28     24     20
#Check C row = 3, #Nodes = 5
   2600   2640   2400   1880   1080
#Check A row = 4, #Nodes = 5
     50     45     40     35     30
#Check C row = 4, #Nodes = 5
   3750   3800   3450   2700   1550
Brodcasted matrix B:     10     12     12     10     06
     15     16     15     12     07
     20     20     18     14     08
     25     24     21     16     09
```

```
     30     28     24     18     10
Gathered matrix D:    350    360    330    260    150
    900    920    840    660    380
   1650   1680   1530   1200    690
   2600   2640   2400   1880   1080
   3750   3800   3450   2700   1550
Run time on  5 nodes: 0.000059
```

Comments in the code below:

```
#include <iostream>
#include <mpi.h>
using namespace std;
#define N 3

double A[N];
double B[N][N];
double C[N];
double D[N][N];

static void init_matrix(void){ //used to initialise matrix B in main
int i, j;
for (i = 0; i < N; i++){
    for (j = 0; j < N; j++){
        B[i][j] = (j+i+2)*(N-j);
                            }
                        }
}

static void init_row(int i){  //used to assign row to each rank process i
int  j;
    for (j = 0; j < N; j++){
        A[j] = (N-j+i+1)*(i+1);
                            }
}


static void print_matrix(double M[N][N]){ //used to print arbitrary matrix M
int i, j;
for(i = 0; i < N; i++){
    for(j = 0; j < N; j++){
        printf("%7.2d", int(M[i][j]));
                    }
        printf("\n");
        }
}

static void print_row(double M[N]){  // used to print arbitrary row M
int j;
    for(j = 0; j < N; j++){
```

```
        printf("%7.2d", int(M[j]));
                    }
        printf("\n");
}

int main(){

    int rank, nproc;
    double start_time, end_time;

    // Start parallel sequence
    MPI_Init(NULL, NULL);
    MPI_Comm comm;
    comm  = MPI_COMM_WORLD;
    MPI_Comm_size(comm, &nproc);
    MPI_Comm_rank(comm, &rank);

    init_matrix();
    start_time = MPI_Wtime();
    MPI_Bcast(B, N, MPI_DOUBLE, 0, comm); //Broadcast matrix B to processes
    MPI_Barrier(comm);  //use Barrier to make sure each process gets B

    init_row(rank);
    printf("#Check A row = %d, #Nodes = %d\n", rank, nproc);
    // Matrix (row-vector) multiplication
    print_row(A);
    for(int j=0; j<N; j++){
           C[j] = 0;
           for(int k=0; k<N; k++){
              C[j] += A[k]*B[k][j]; }
        }
    printf("#Check C row = %d, #Nodes = %d\n", rank, nproc);
    print_row(C);  //Print row C to check if correctly assigned

  // Gather rows from individual processes
     MPI_Gather(C, N, MPI_DOUBLE, D, N, MPI_DOUBLE, 0, comm);
     end_time = MPI_Wtime();
     MPI_Barrier(comm);
     MPI_Finalize();

  // Display Gathered matrix and results
   if (rank == 0){
        printf("Brodcasted matrix B:");
        print_matrix(B);
        printf("Gathered matrix D:");
        print_matrix(D);
        printf("Run time on %2d nodes: %f\n", nproc, end_time-start_time);
}

     return 0;
```