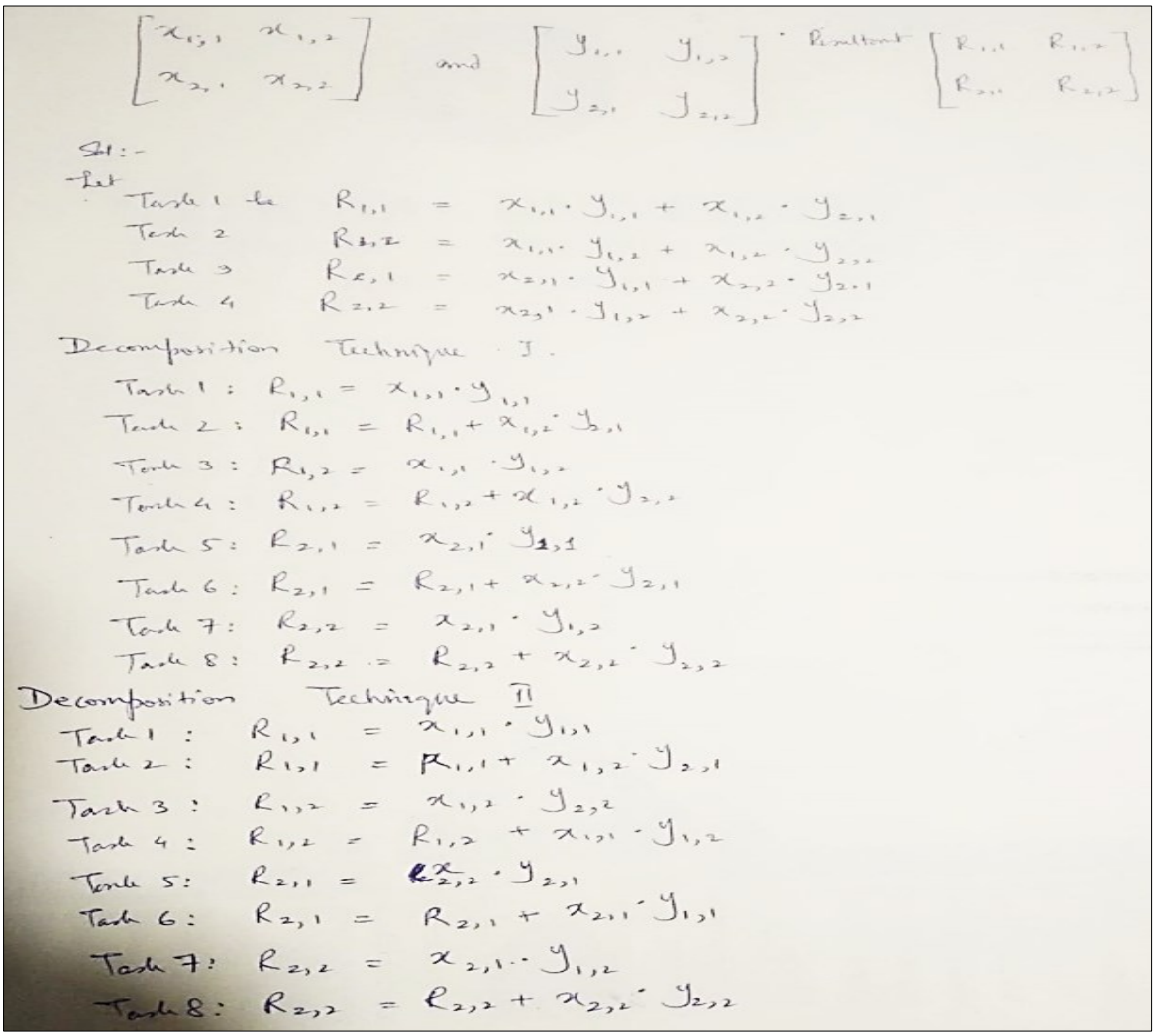
Time: 60 Minutes. **Max Marks**: 15 Points (15 Marks)

Question 1: [Marks = 3+3+2+2]

a)

```
1   procedure RECURSIVE_MIN (A, n)
2   begin
3      if (n = 1) then
4         min := A[0];
5   else
6      lmin := RECURSIVE_MIN (A, n/2);
7      rmin := RECURSIVE_MIN (&(A[n/2]), n - n/2);
8      if (lmin < rmin) then
9         min := lmin;
10     else
11        min := rmin;
12     endelse;
13  endelse;
14  return min;
15  end RECURSIVE_MIN
```

b) Data Decomposition Technique



c)

Critical path length = 29

Total work done = 42 Average degree of Concurrency = 42/29 = 1.44

d) The CheckpointNode periodically combines the existing checkpoint and journal to create a new checkpoint and an empty journal. The CheckpointNode usually runs on a different host from the NameNode since it has the same memory requirements as the NameNode. It downloads the current checkpoint and journal files from the NameNode, merges them locally, and returns the new checkpoint back to the NameNode.

Question 2: [2+3]

a) If MPI_Send is implemented by blocking until the matching receive has been issued, then neither of the two processes will be able to proceed. This is because process zero (i.e., myrank == 0) will wait until process one issues the matching MPI_Recv (i.e., the one with tag equal to 1), and at the same time process one will wait until process zero performs the matching MPI_Send (i.e., the one with tag equal to 2).

b)

```c
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int rank, size, N = 3;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int a[N][N] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
    int b[N][N] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
    int result[N][N];
    int aa[N], bb[N], rowresult[N];

    // scatter rows of first and seconod matrix to different processes
    MPI_Scatter(a, N * N / size, MPI_INT, aa, N * N / size, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Scatter(b, N * N / size, MPI_INT, bb, N * N / size, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);

    // perform vector addition by all processes
    for (int j = 0; j < N; j++)
        rowresult[i] = aa[j] + bb[j];

    MPI_Gather(rowresult, N * N / size, MPI_INT, result, N * N / size, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);

    if (rank == 0) {
        printf("\n\n%s\n", “Result =”);
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                printf(" %d", result[i][j]);
            }
            printf("\n");
        }
        printf("\n\n");
    }

    MPI_Finalize();
    return 0;
}
```