Verification Analysis:
For verification of sums and sums of squares, please see the screenshots. (Time analysis later)

```
Sum of entries for m = 2000, n = 500, and p = 1 is: 3.60896e+06
Sum of square of entries for m = 2000, n = 500, and p = 1 is: 4.67742e+09
Elapsed time for m = 2000, n = 500, and p = 1 is: 28.5338 seconds


Sum of entries for m = 2000, n = 500, and p = 4 is: 3.60896e+06
Sum of square of entries for m = 2000, n = 500, and p = 4 is: 4.67742e+09
Elapsed time for m = 2000, n = 500, and p = 4 is: 7.28766 seconds


Sum of entries for m = 2000, n = 500, and p = 16 is: 3.60896e+06
Sum of square of entries for m = 2000, n = 500, and p = 16 is: 4.67742e+09
Elapsed time for m = 2000, n = 500, and p = 16 is: 5.54674 seconds


Sum of entries for m = 2000, n = 500, and p = 36 is: 3.60896e+06
Sum of square of entries for m = 2000, n = 500, and p = 36 is: 4.67742e+09
Elapsed time for m = 2000, n = 500, and p = 36 is: 0.851567 seconds
```

I verify that for m = 2000, n = 500, sum = 3.609e+06, sum of squares = 4.677e+09.

```
Sum of entries for m = 1000, n = 4000, and p = 1 is: 6.53689e+06
Sum of square of entries for m = 1000, n = 4000, and p = 1 is: 3.71512e+10
Elapsed time for m = 1000, n = 4000, and p = 1 is: 114.463 seconds


Sum of entries for m = 1000, n = 4000, and p = 4 is: 6.53689e+06
Sum of square of entries for m = 1000, n = 4000, and p = 4 is: 3.71512e+10
Elapsed time for m = 1000, n = 4000, and p = 4 is: 29.2319 seconds


Sum of entries for m = 1000, n = 4000, and p = 16 is: 6.53689e+06
Sum of square of entries for m = 1000, n = 4000, and p = 16 is: 3.71512e+10
Elapsed time for m = 1000, n = 4000, and p = 16 is: 22.2158 seconds


Sum of entries for m = 1000, n = 4000, and p = 36 is: 6.53689e+06
Sum of square of entries for m = 1000, n = 4000, and p = 36 is: 3.71512e+10
Elapsed time for m = 1000, n = 4000, and p = 36 is: 3.35978 seconds
```

I verify that for m = 1000, n = 4000, sum = 6.537e+06, sum of squares = 3.715e+10.

Technique Analysis:
Brief description of how I decompose the matrix, and how different parts communicate with each other:
Let me take m = 11, n = 9, and p = 4 as an example.
With careful calculation of size and index, I decompose the original matrix into 4 parts:
(1) upper-left (row 0-5, col 0-4, rank 0, row id 0, col id 0);
(2) upper-right (row 0-5, col 5-8, rank 1, row id 0, col id 1);
(3) lower-left (row 6-10, col 0-4, rank 2, row id 1, col id 0);
(4) lower-right (row 6-10, col 5-8, rank 3, row id 1, col id 1).

Then for the initialization for each localMatrix, let it be new int[rowSize+2][colSize+2].
Plus 2 represents ghost rows or cols, which means that we need to store the border rows and columns from neighbor matrixes, where the top row stores the original bottom row from the upper matrix, bottom row stores the original top row from the lower matrix, leftmost row stores the original rightmost row from the left matrix, rightmost row stores the original leftmost row from the right matrix. Four corner cells are left as empty, and if the localMatrix is located on the border, then corresponding ghost rows or columns are left as empty.

The reason is that for each cell localMatrix[i][j], we need to calculate localMatrix[i-1][j], localMatrix[i+1][j], localMatrix[i][j-1], localMatrix[i][j+1], for border rows and columns, we need to get the values from the neighboring matrixes.

Regarding the communication, it's just MPI_SEND(&original top row, colSize, rank - 2), MPI_RECV(&ghost top row, colSize, rank - 2), and similar for other 3 directions.

Time Analysis:
We observe that in most cases, elapsed times are almost inversely proportional to number of processors (slope is 4), except for from 4 processors to 16 processors, which is not 1/4 times in elapsed time.
In addition, elapsed times are also almost to proportional to size of the matrix (slope is 4).
Therefore, we conclude that the time represents perfect speedup and scaling.

Please refer to the attached plot for communication among localMatrixes.