```c
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <mpi.h>

int n, p;

int main(int argc, char **argv) {
  int myn, myrank;
  double *a, *b, *c, *allB, start, sum, *allC, sumdiag;
  int i, j, k;

  n = atoi(argv[1]);
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD,&p);
  MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
  myn = n/p;
  a = malloc(myn*n*sizeof(double));
  b = malloc(myn*n*sizeof(double));
  c = malloc(myn*n*sizeof(double));
  allB = malloc(n*n*sizeof(double));
  for(i=0; i<myn*n; i++) {
      a[i] = 1.;
      b[i] = 2.;
  }
  MPI_Barrier(MPI_COMM_WORLD);
  if(myrank==0)
    start = MPI_Wtime();
  for(i=0; i<p; i++)
    MPI_Gather(b, myn*n, MPI_DOUBLE, allB, myn*n, MPI_DOUBLE,
               i, MPI_COMM_WORLD);
  for(i=0; i<myn; i++)
    for(j=0; j<n; j++) {
      sum = 0.;
      for(k=0; k<n; k++)
        sum += a[i*n+k]*allB[k*n+j];
      c[i*n+j] = sum;
    }
  free(allB);
  MPI_Barrier(MPI_COMM_WORLD);
  if(myrank==0)
    printf("It took %f seconds to multiply 2 %dx%d matrices.\n",
           MPI_Wtime()-start, n, n);
  if(myrank==0)
    allC = malloc(n*n*sizeof(double));
  MPI_Gather(c, myn*n, MPI_DOUBLE, allC, myn*n, MPI_DOUBLE,
             0, MPI_COMM_WORLD);
  if(myrank==0) {
    for(i=0, sumdiag=0.; i<n; i++)
      sumdiag += allC[i*n+i];
    printf("The trace of the resulting matrix is %f\n", sumdiag);
  }
  if(myrank==0)
    free(allC);
  MPI_Finalize();
```

```
    free(a);
    free(b);
    free(c);
}
```

It is assumed that the user via an external parameter supplies the value for n and this value is a multiple of the total number of processes executing the program, `p`. It is also assumed that process zero (`myrank`=0) runs on the node with I/O available. Each process has the following arrays on heap:

- `a`, `b`, and `c` to store its horizontal $myn \times n$ slice of matrices *A*, *B*, and *C* respectively;
- `allB` to store the full matrix *B*. This array is deallocated as soon as the process has computed all elements of its *C* slice.

The array `allC` is allocated on process zero after the parallel multiplication itself has been complete. This array is used as a receive buffer by the collective operation that gathers all slices of the resulting matrix *C* on process zero. After the matrix *C* is gathered on process zero in the array `allC`, this process computes the trace of the matrix and output it (just to let the user check correctness of the computations).

By the way, the loop

```
for(i=0; i<p; i++)
  MPI_Gather(b, myn*n, MPI_DOUBLE, allB, myn*n, MPI_DOUBLE,
             i, MPI_COMM_WORLD);
```

resulting in all processes to receive the matrix *B*, can be replaced by a single call of the form

```
MPI_Allgather(b, myn*n, MPI_DOUBLE, allB, myn*n, MPI_DOUBLE,
              MPI_COMM_WORLD);.
```

`MPI_Allgather` is an example of advanced variation of the basic gather operation. It is assumed that MPI may implement `MPI_Allgather` more efficiently than via multiple calls to `MPI_Gather`.