```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>

#define TAG 123

int main(int argc, char *argv[])
{
    int n, ii, jj, i, j, k;
    int *a, *b, *c;
    int *aa, *bb, *cc;
    int mSize;
    int rankOfProcessor;
    int numberOfProcessors;
    int masterProcessor = 0;
    int start, end;
    int iNum, iStart, iEnd;
    char *buf;
    int bufSize;

    MPI_Status status;
    MPI_Init ( &argc, &argv );
    MPI_Comm_size ( MPI_COMM_WORLD, &numberOfProcessors );
    MPI_Comm_rank ( MPI_COMM_WORLD, &rankOfProcessor );

    if(rankOfProcessor == masterProcessor)
        printf("Number of Processors = %d\n", numberOfProcessors);

    for(n=1000; n<=5000; n+=1000)
    {
        mSize = sizeof(int) * n * n;
        a = (int *) malloc(mSize);
        b = (int *) malloc(mSize);
        c = (int *) malloc(mSize);

        memset(c, 0, mSize);

        if(rankOfProcessor == masterProcessor)
        {
            srand(10);

            for(i=0; i<n*n; ++i)
            {
                a[i] = rand() % 10;
                b[i] = rand() % 10;
            }

            start = clock();
        }

        // SEND MATRIX A & B

        if(rankOfProcessor == masterProcessor)
            bufSize = mSize * numberOfProcessors;
        else
            bufSize = mSize * 2;
        buf = malloc(bufSize);
        MPI_Buffer_attach( buf, bufSize );

        MPI_Barrier(MPI_COMM_WORLD);

        MPI_Bcast(a, n*n, MPI_INT, masterProcessor, MPI_COMM_WORLD);
        MPI_Bcast(b, n*n, MPI_INT, masterProcessor, MPI_COMM_WORLD);

        // BEGINNING OF MATRIX MULTIPLICATION

        iNum = (n/40 + numberOfProcessors - 1) / numberOfProcessors;
        iStart = rankOfProcessor * iNum;
        iEnd = (rankOfProcessor+1) * iNum - 1;
        if(n/40-1 < iEnd)
            iEnd = n/40-1;
        //printf("I will calculate the rows (%d - %d)\n", iStart, iEnd);

        for(ii=iStart; ii<=iEnd; ++ii)
        {
            for(jj=0; jj<n/8; ++jj)
            {
                for(i=40*ii; i<40*(ii+1); ++i)
                {
                    aa = &a[i * n + jj*8];
                    for(j=jj*8; j<(jj+1)*8; ++j)
                    {
                        cc = &c[i * n];
                        bb = &b[j * n];

                        for(k=0; k<n; ++k)
                            *cc++ += *aa * *bb++;
                        ++aa;
                    }
                }
            }
        }

        // END OF MATRIX MULTIPLICATION

        if(rankOfProcessor == masterProcessor)
        {
            for(i=1; i<numberOfProcessors; ++i)
            {
                iStart = i * iNum;
                iEnd = (i+1) * iNum - 1;
                if(n/40-1 < iEnd)
                    iEnd = n/40-1;
                if(iEnd >= iStart)
                    MPI_Recv(c + n*40*iStart, (iEnd-iStart+1)*n*40, MPI_INT, i, TAG,       ↙
    MPI_COMM_WORLD, &status);
            }
        }
        else
        {
            if(iEnd >= iStart)
                MPI_Bsend(c + n*40*iStart, (iEnd-iStart+1)*n*40, MPI_INT, 0, TAG,          ↙
    MPI_COMM_WORLD);
        }

        MPI_Buffer_detach(buf, &bufSize);

        // MATRIX C IS RECEIVED

        if(rankOfProcessor == masterProcessor)
        {
            end = clock();
            printf("  (%d x %d) -> duration = %d ms\n", n, n, end-start);
        }

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

    MPI_Finalize();

    return 1;
}
```