Matrix Mutiplication

-AIM: Matrix Multiplication using MPI

> Cpu Specs:

Memory	5.8 GiB
Processor	Intel® Core™ i3-4130 CPU @ 3.40GHz × 4
Graphics	Intel® Haswell x86/MMX/SSE2

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
#include <unistd.h>
int size = 2;
void print(int a[][size]){
    int i,j;
    for(i=0;i<size;i++){
            printf("\n");
            for(j=0;j<size;j++)
                    printf(" %d\t",a[i][j]);
    }
    printf("\n");
}
int main() {
    double start,end;
    int i, j, k, A[size][size], B[size][size], C[size][size];
    int *recv = (int*)malloc(size*sizeof(int));
    int *sum = (int*)malloc(size*sizeof(int));
    MPI_Init(NULL, NULL);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    if(world_size < size) {
            printf("world_size < size\n");
            exit(0);
    }
    if((world_size%size)!=0) {
            printf("(world_size mod size)!=0\n");
            exit(0);
    }
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    if(world_rank == 0){
            for(i=0;i<size;i++){
                    for(j=0;j<size;j++){
                            A[i][j]=i+j;
                            B[i][j]=i+j;
                    }
            }
            start = MPI_Wtime();
    }
    MPI_Bcast(B, (size*size), MPI_INT, 0, MPI_COMM_WORLD);
    for(i = 0; i < (size/world_size); i++) {
            MPI_Scatter(A+i*world_size,size,MPI_INT,recv,size,MPI_INT,0,MPI_COMM_WORLD);
            for (j = 0; j < size; ++j)
            {       sum[j] = 0;
                    for (k = 0; k < size; ++k)
                    {
                            sum[j] += recv[k]*B[k][j];
```

```
			}
		}
		MPI_Gather(sum,size,MPI_INT,C+i*world_size,size,MPI_INT,0,MPI_COMM_WORLD);
	}
	if(world_rank == 0){
		end = MPI_Wtime();
		printf("Time:%lf\n", (end - start));
		//print(C);
	}
	MPI_Finalize();
}
```

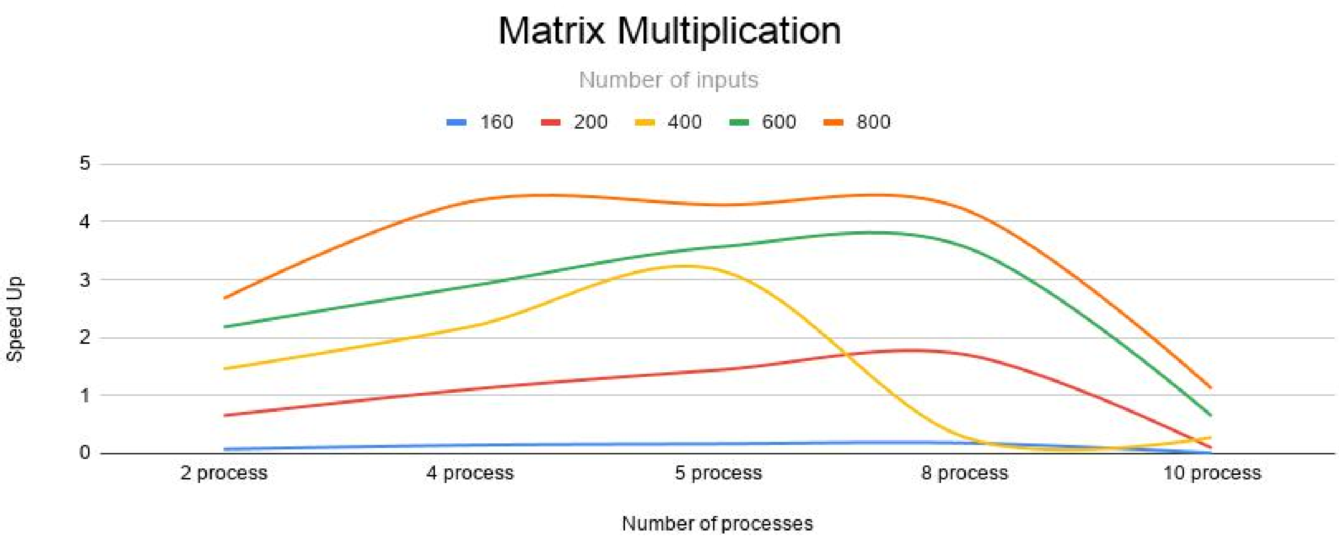
-Explaination:

- In above code we send each N/P rows to each P processes in each iteration using scatter until all rows are processed of first matrix and Broadcast matrix B to all processes the each process compute their result and it is gathered by master process using gather.
- Mainly in this parallel code each process will compute single-single row if total number of rows equal to total number of processes otherwise eacah process will compute N/P rows.

- Analysis By changing size of matrix and number of process

Process\size	160	200	400	600	800
2 process	0.07520641153	0.6553446964	1.460715887	2.184624733	2.674831157
4 process	0.1455624382	1.109008037	2.191616466	2.892882687	4.352043134
5 process	0.1692633618	1.440828135	3.175747436	3.566763317	4.294471432
8 process	0.1827712418	1.711355705	0.2796338761	3.573013587	4.215954601
10 process	0.009625333448	0.09467009222	0.2772577162	0.6467117955	1.120926105

Speedup -



-Conclusion:

- We can see we can get better scalability if we increase number of inputs with respect to number of processes, We can see that for N=600 and N=800.
- But after number of process greater than 8 speedup is decreasing because we have only total 8 cores available.
- For smaller amout of input there is no major difference in speedup that is observed for N=160.