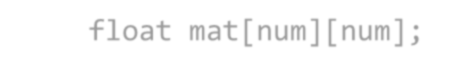
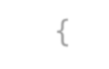
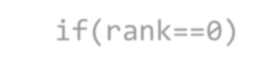
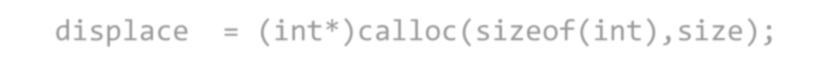
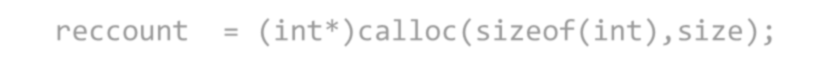
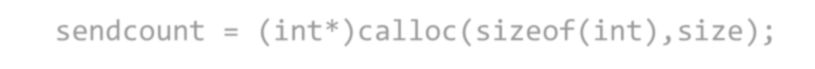
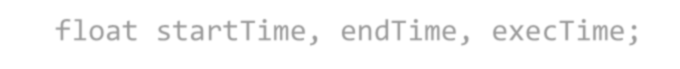
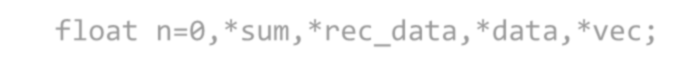
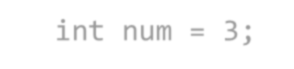
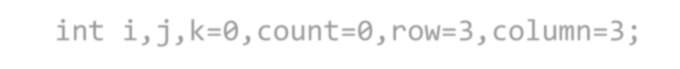
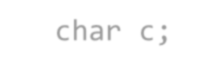
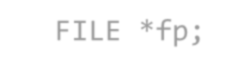
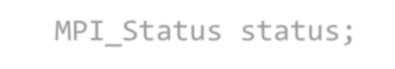
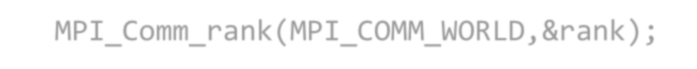
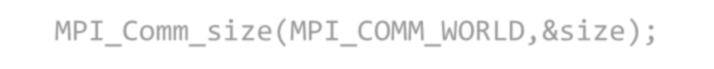
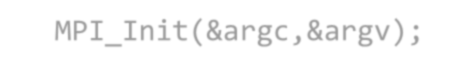
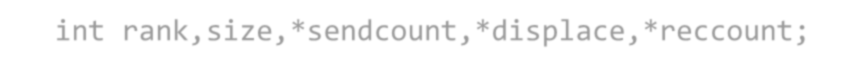
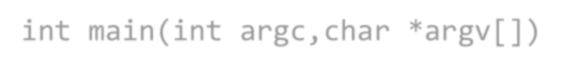
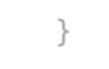
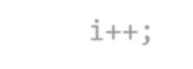
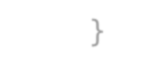
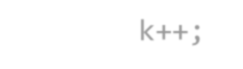
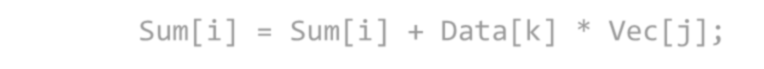
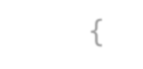
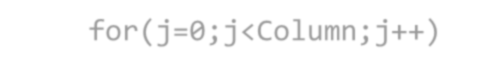
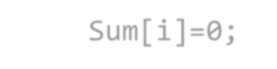
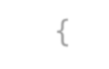
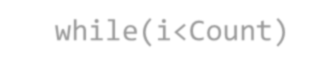
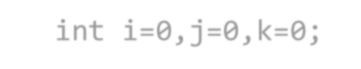
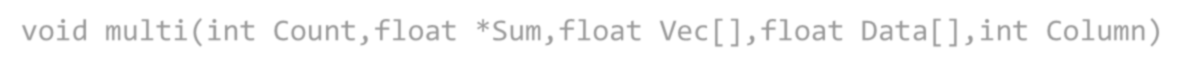
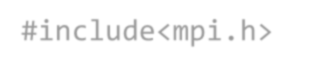
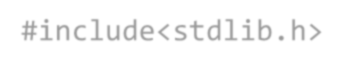
MATRIX VECTOR MULTIPLICATION

SOURCE CODE:



#include<stdio.h>

#include<stdlib.h> #include<mpi.h>

void multi(int Count,float \*Sum,float Vec[],float Data[],int Column)

{

int i=0,j=0,k=0; while(i<Count)

{

Sum[i]=0; for(j=0;j<Column;j++)

{

Sum[i] = Sum[i] + Data[k] \* Vec[j]; k++;

} i++;

}

}

int main(int argc,char \*argv[])

{

int rank,size,\*sendcount,\*displace,\*reccount; MPI\_Init(&argc,&argv); MPI\_Comm\_size(MPI\_COMM\_WORLD,&size); MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); MPI\_Status status;

FILE \*fp; char c;

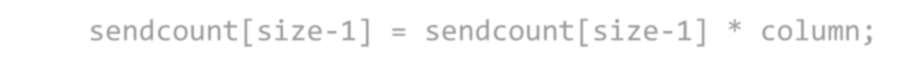
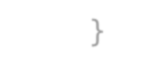
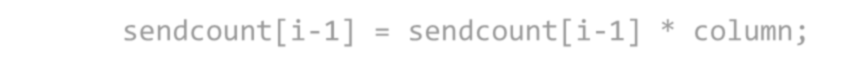
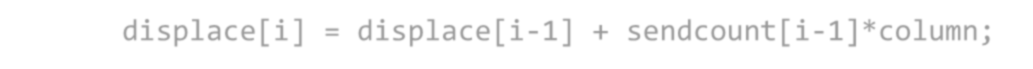
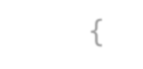
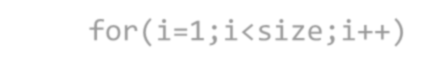
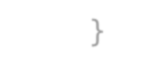
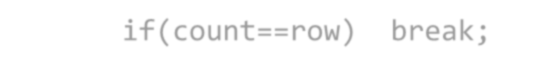
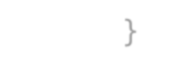
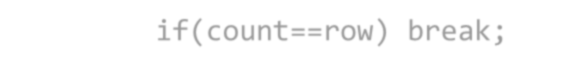
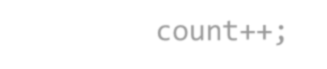
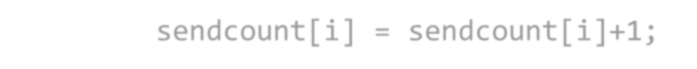
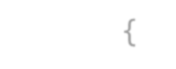
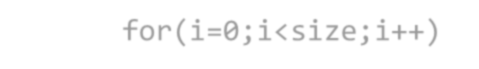
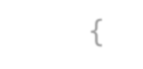
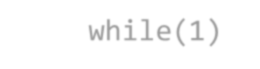
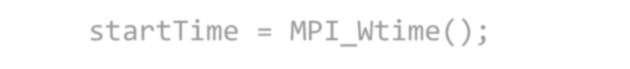
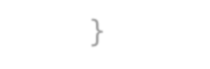
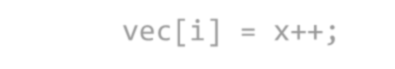
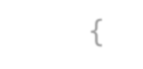
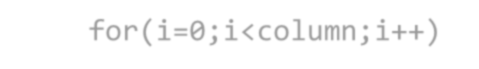
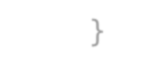
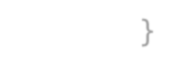
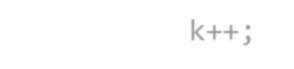
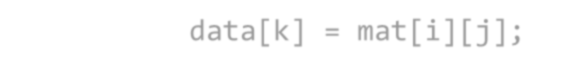
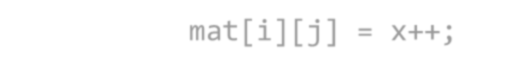
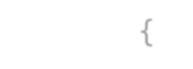
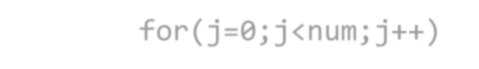
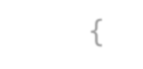
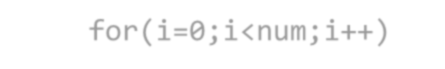
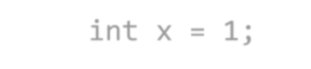
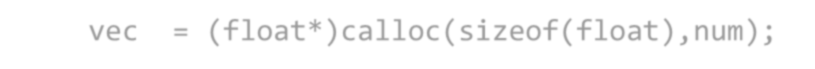
int i,j,k=0,count=0,row=3,column=3; int num = 3;

float n=0,\*sum,\*rec\_data,\*data,\*vec; float startTime, endTime, execTime;

sendcount = (int\*)calloc(sizeof(int),size); reccount = (int\*)calloc(sizeof(int),size); displace = (int\*)calloc(sizeof(int),size); if(rank==0)

{

float mat[num][num];



data = (float\*)calloc(sizeof(float),num\*num);

vec = (float\*)calloc(sizeof(float),num); int x = 1;

for(i=0;i<num;i++)

{

for(j=0;j<num;j++)

{

mat[i][j] = x++;

data[k] = mat[i][j]; k++;

}

}

for(i=0;i<column;i++)

{

vec[i] = x++;

}

startTime = MPI\_Wtime(); while(1)

{

for(i=0;i<size;i++)

{

sendcount[i] = sendcount[i]+1; count++;

if(count==row) break;

}

if(count==row) break;

}

for(i=1;i<size;i++)

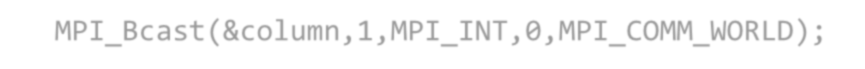
{

displace[i] = displace[i-1] + sendcount[i-1]\*column; sendcount[i-1] = sendcount[i-1] \* column;

}

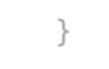
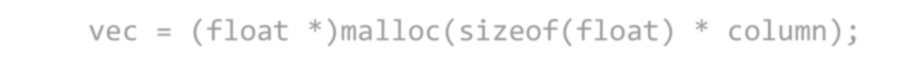
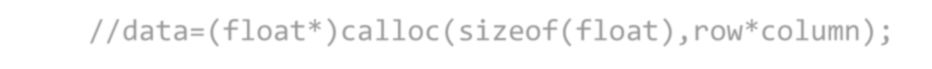
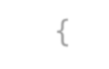
sendcount[size-1] = sendcount[size-1] \* column;

}



MPI\_Bcast(&row,1,MPI\_INT,0,MPI\_COMM\_WORLD);

MPI\_Bcast(&column,1,MPI\_INT,0,MPI\_COMM\_WORLD);

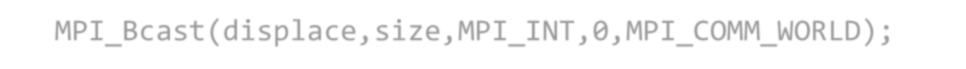
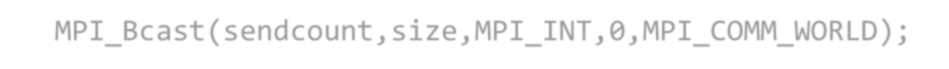


if(rank!=0)

{

//data=(float\*)calloc(sizeof(float),row\*column); vec = (float \*)malloc(sizeof(float) \* column);

}



MPI\_Bcast(vec,column,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

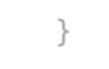
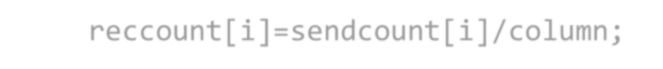
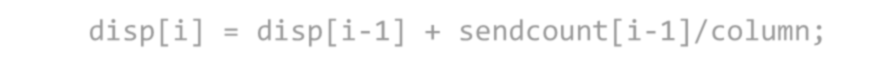
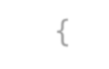
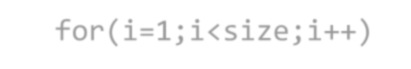
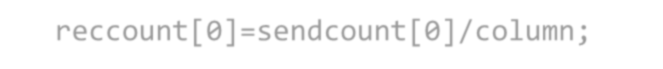
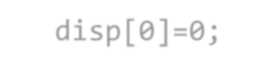
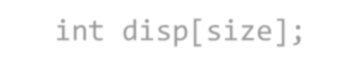
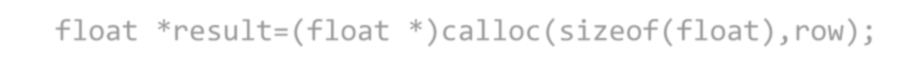
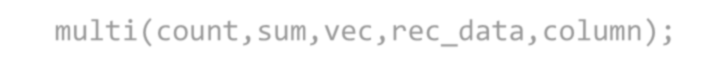
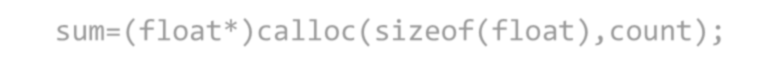
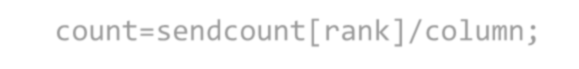
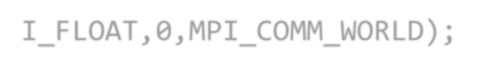
MPI\_Bcast(sendcount,size,MPI\_INT,0,MPI\_COMM\_WORLD); MPI\_Bcast(displace,size,MPI\_INT,0,MPI\_COMM\_WORLD);



rec\_data=(float\*)calloc(sizeof(float),sendcount[rank]);



//MPI\_Bcast(data,row\*column,MPI\_INT,0,MPI\_COMM\_WORLD);



MPI\_Scatterv(data,sendcount,displace,MPI\_FLOAT,rec\_data,sendcount[rank],MP

I\_FLOAT,0,MPI\_COMM\_WORLD);

count=sendcount[rank]/column; sum=(float\*)calloc(sizeof(float),count); multi(count,sum,vec,rec\_data,column);

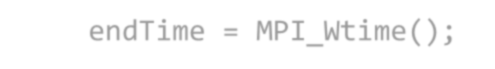
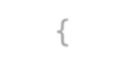
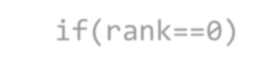
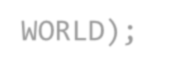
float \*result=(float \*)calloc(sizeof(float),row); int disp[size];

disp[0]=0; reccount[0]=sendcount[0]/column; for(i=1;i<size;i++)

{

disp[i] = disp[i-1] + sendcount[i-1]/column; reccount[i]=sendcount[i]/column;

}



MPI\_Gatherv(sum,count,MPI\_FLOAT,result,reccount,disp,MPI\_FLOAT,0,MPI\_COMM\_

WORLD);

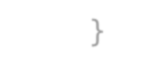
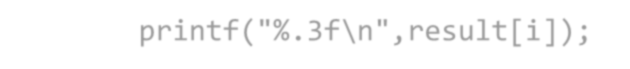
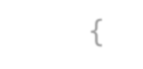
if(rank==0)

{

endTime = MPI\_Wtime();



printf("\nMatrix Vector Multiplication is:\n");

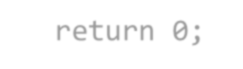
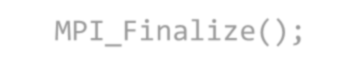
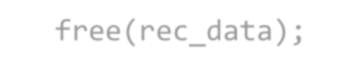
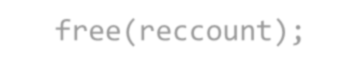
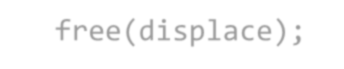
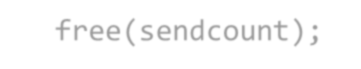
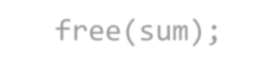
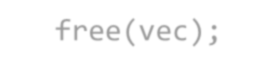
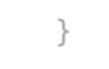


for(i=0;i<row;i++)

{

printf("%.3f\n",result[i]);

}



printf("\nExecution Time is %f\n",endTime-startTime);

}

free(vec); free(sum); free(sendcount); free(displace); free(reccount); free(rec\_data); MPI\_Finalize();

return 0;

}