MPI 矩阵-向量乘法

一、学生信息

姓名：柳建国
学号：2022Z8017782089
专业：电子信息
所部：数字所

二、问题描述

对于矩阵 A（如图 1）和向量 $x = (1,2,3,1,2,3,\ldots)$，计算向量 $y = A \times x$（如图 2）。

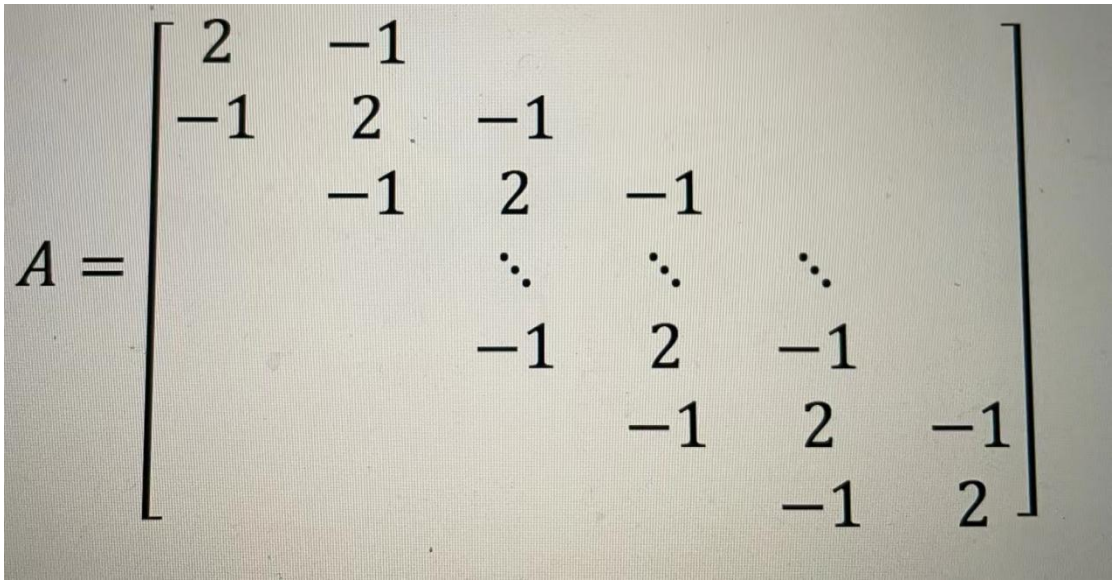

图 1 矩阵 A 形式

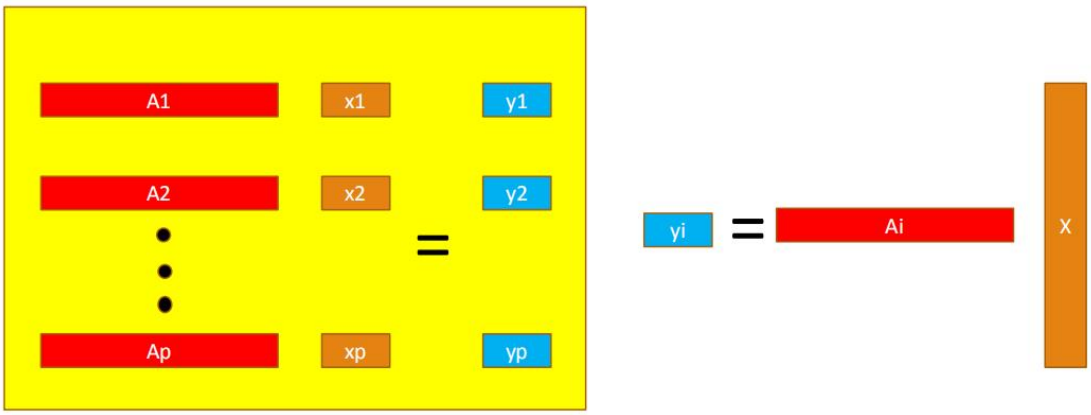

图 2 向量乘示意图

三、测试环境（系统，CPU，核心）

测试环境配置见表 1。

表 1 运行环境配置

配置	参数
系统	Ubuntu 18.04
CPU	Inter(R) Core(TM)i5-10505 CPU@5.20GHz 3.20Ghz
核心	虚拟机配置 4 核心
内存大小	8G

四、MPI 程序的性能评估

1. 运行时间

表 2 不同数量进程和各种大小矩阵下，程序运行时间

Comm_sz	Order of Matrix				
	1024	2048	4096	8192	16384
1	0.01044625	0.01382025	0.04634375	0.178577	0.716706
2	0.0063685	0.00816425	0.02244475	0.08978375	0.5355865
4	0.002643	0.004497	0.0127355	0.04668875	0.1850915

2. 加速比

表 3 不同数量进程和各种大小矩阵下，程序运行加速比

Comm_sz	Order of Matrix				
	1024	2048	4096	8192	16384
1	1.00	1.00	1.00	1.00	1.00
2	1.64	1.69	2.06	1.99	1.34
4	3.95	3.07	3.64	3.82	3.87

3. 效率

表 4 不同数量进程和各种大小矩阵下，程序运行效率

Comm_sz	Order of Matrix				
	1024	2048	4096	8192	16384
1	1.00	1.00	1.00	1.00	1.00
2	0.82	0.85	1.03	0.99	0.82

4	0.99	0.77	0.91	0.96	0.98

4. 可扩展性

如果一个技术可以处理规模不断增加的问题，那么它就是可扩展的。对于并行程序而言，可扩展性有明确定义。

假设我们运行一个并行程序，固定进程或线程数目、固定问题规模，得到一个效率值 E。现在我们增加程序所用的进程或线程数目，如果在问题规模也同比例增加情况下，该程序效率值一直都是 E，那么我们就称该程序是可扩展的。

如果在增加进程或线程个数时，可以维持固定效率，却不增加问题规模，那么程序成为强可扩展(strongly scalable)。如果在增加进程或线程个数时，只有以相同倍率增加问题规模才能保持效率值，那么程序就称为弱可扩展(weakly scalable)。

在不考虑进程间通信的情况下，通过以上实验结果，可以看到，随着进程数量增加，程序执行的时间在线性减小，所以该程序算法是强可扩展性的。

五、源代码

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

//Matrix number N*N
#define N 1024

//claim function
int ** makeMatrixA(int matrixsize);
int * makeMatrixx(int matrixsize);

//this is main function
int main(int argc, char *argv[]){
	// init the MPI
	int rank, size;
	MPI_Init(&argc, &argv);	//MPI Initialize
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);	//obtain the rank id
	MPI_Comm_size(MPI_COMM_WORLD, &size);	//obtain the number of processes

	if(rank==0){
		printf("The size is:%d\n", size);
	}

	//make Matrix
```

```
int ** MatrixA = makeMatrixA(N);
/* ===test for MatrixA===
if(rank ==0){
	printf("MatrixA is: \n");
	for(int i=0; i<N; i++){
		for(int j=0; j<N; j++){
			printf("%d, ",MatrixA[i][j]);
		}
		printf("\n");
	}
}
*/
int * Matrixx = makeMatrixx(N);
/* ===Test for Matrixx===
if(rank ==0 ){
	printf("Matrixx is:\n ");
	for(int i=0; i<N; i++){
		printf("%d ", Matrixx[i]);
	}
	printf("\n");
}
*/

MPI_Barrier(MPI_COMM_WORLD);
//start time
double local_start, local_end;
local_start = MPI_Wtime();

//each process calculate the result
int * y = (int *)malloc(sizeof(int)*N);
int * localy = (int *)malloc(sizeof(int)*N/size);
int localN = N/size;
for(int i=0; i<localN; i++){
	localy[i]=0;
	for(int j=0; j<N; j++){
		localy[i] += MatrixA[rank*localN+i][j] * Matrixx[j];
		/* ===Test for localy===
		if(rank == 0){
			printf("%d ", MatrixA[rank*localN+i][j]*Matrixx[j]);
		}
		*/
	}
	//printf("\n");
}
```

```
	/* ===Test for localy===
	if(rank ==0){
		printf("\n process 0 local y is:\n");
		for(int i=0; i<localN; i++){
			printf("%d ", localy[i]);
		}
		printf("\n");
	}
	*/

	MPI_Barrier(MPI_COMM_WORLD);
	//end time
	local_end = MPI_Wtime();

	//get result together from all process
	MPI_Gather(localy, localN, MPI_INT, y, localN, MPI_INT, 0, MPI_COMM_WORLD);

	//print result
	if(rank == 0){
		printf("Execute time is %lf.\n", local_end-local_start);
		printf("The result is:");
		for(int i=0; i<N; i++){
			printf("%d ", y[i]);
		}
		printf("\n");
	}

	// free memory
	free(MatrixA);
	free(Matrixx);
	free(localy);
	free(y);

	MPI_Finalize(); //Finalize MPI

}
int ** makeMatrixA(int matrixsize){
	int **MatrixA = (int **)malloc(sizeof(int *)*matrixsize);
	for(int i=0; i<N; i++){
		MatrixA[i] = (int *)malloc(sizeof(int)*matrixsize);
	}
	for(int i=0; i<matrixsize; i++){
		for(int j=0; j<matrixsize; j++){
			if(j==(i-1)||j==(i+1)){
```

```
                        MatrixA[i][j]=-1;
                }
                else if(j==i){
                        MatrixA[i][j]=2;
                }
                else{
                        MatrixA[i][j]=0;
                }

        }
    }
    return MatrixA;
}
int * makeMatrixx(int matrixsize){
    int * Matrixx=(int *)malloc(sizeof(int)*matrixsize);
    for(int i=0; i<matrixsize; i++){
        if(i%3 == 0){
                Matrixx[i] = 1;
        }
        else if(i%3 ==1){
                Matrixx[i] = 2;
        }
        else{
                Matrixx[i] = 3;
        }
    }
    return Matrixx;
}
```