lab1-基于MPI的并行矩阵乘法

实验要求

使用MPI点对点通信方式实现并行通用矩阵乘法(MPI-v1)，并通过实验分析不同进程数量、矩阵规模时该实现的性能。

输入： m, n, k三个整数，每个整数的取值范围均为[128, 2048]

问题描述：随机生成$m \times n$的矩阵A及$n \times k$的矩阵B，并对这两个矩阵进行矩阵乘法运算，得到矩阵C.

输出： A, B, C三个矩阵，及矩阵计算所消耗的时间 。

要求：1. 使用MPI点对点通信实现并行矩阵乘法，调整并记录不同线程数量（1~16）及矩阵规模（128~2048）下的时间开销，填写下页表格，并分析其性能。

关键代码

矩阵初始化

这里首先初始化矩阵

```
// 初始化矩阵
void init_Mat(int row, int col,double* mat){
    for (int m=0;m<row;m++){
        for (int n=0;n<col;n++){
            mat[m*col+n]=(double)(rand()%1000/10.0);
        }
    }
}
```

然后通过解析命令行得到矩阵的维度并动态分配矩阵和结果矩阵的空间：

```
    // 解析命令行参数得到的矩阵维度
    int M=atoi(argv[1]);
    int N=atoi(argv[2]);
    int K=atoi(argv[3]);
    // 动态分配存储矩阵和结果矩阵的空间
    double *b= new double [ N* K ];
    double *result = new double [ M * K ];
    double *a=NULL,*c=NULL;
    int pid, process_num, line;
```

MPI初始化

调用`MPI_Init`来初始化MPI环境，并获取当前进程的ID和总进程数。

```
    // 初始化MPI环境
    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD,&pid);     // 获取当前进程的ID
    MPI_Comm_size(MPI_COMM_WORLD,&process_num);     // 获取进程总数
```

主进程

主进程使用`MPI_Send`将矩阵A和B分发到各个子进程并通过`MPI_Recv`对子进程的计算结果进行接收，然后再根据从子进程的计算结果负责矩阵C的计算：

```
        // 发送矩阵A和 B到子进程
        for (int i=1;i<process_num;i++){
            MPI_Send(b,N*K,MPI_DOUBLE,i,0,MPI_COMM_WORLD);
        }
        for (int i=1;i<process_num;i++){
            MPI_Send(a+(i-1)*line*N,N*line,MPI_DOUBLE,i,1,MPI_COMM_WORLD);
        }

        // 接收子进程的计算结果
        for (int i=1;i<process_num;i++){

MPI_Recv(result,line*K,MPI_DOUBLE,i,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
            for(int l=0;l<line;l++){
                for(int k=0;k<K;k++){
                    c[((i-1)*line+l)*K+k]=result[l*K+k];
                }
            }
        }

        // 计算矩阵C
        for (int i=(process_num-1)*line;i<M;i++){
            for (int j=0;j<K;j++){
                double tmp=0;
                for (int k=0;k<N;k++)
                    tmp += a[i*N+k]*b[k*K+j];
                c[i*K+j] = tmp;
            }
        }
```

子进程

每个子进程通过`MPI_Recv`接收到数据后，计算分配给它的矩阵A的行与矩阵B的乘积，并将结果通过`MPI_Send`发送回主进程。

```
    // 子进程
    else{
        // 动态分配存储接收矩阵B和A的空间
        double* temp = new double [ N * line ];

        // 接收矩阵B和A的行
```

```
        MPI_Recv(b,N*K,MPI_DOUBLE,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        MPI_Recv(temp,N*line,MPI_DOUBLE,0,1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);

        // 计算矩阵乘法
        for(int i=0;i<line;i++){
            for(int j=0;j<N;j++){
                double tmp=0;
                for(int k=0;k<N;k++)
                    tmp += temp[i*N+k]*b[k*K+j];
                result[i*K+j] = tmp;
            }
        }
        // 发送计算结果到主进程
        MPI_Send(result, line*K, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
```

编译运行

由于之前实验0没有安装对于MPI的编译器，这里需要安装`openmpi-bin`：

```
sudo apt-get openmpi-bin
```

然后编译文件：

```
mpic++ lab1.cpp -o lab1
```

使用`mpirun`运行文件，`-np`后的数表示进程数目，后面输入的三个数分别为三个矩阵的三个维度：M、N、K：

```
mpirun -n 4 ./lab1 128 128 128
```

运行结果如下：

```
(d2l) liuglin@WIN-6G3ESV0SHI5:/mnt/d/Paper/Communication-theory/并行程序设计与算法/实验作业/lab1$ mpirun -n 4
./lab1 128 128 128
using time:0.0041628
```

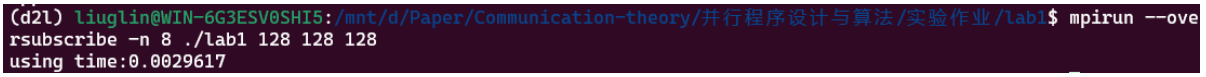
如果指定的进程数过多，数量超过了Open MPI在系统中分配的slots数目，会有如下报错：

```
(d2l) liuglin@WIN-6G3ESV0SHI5:/mnt/d/Paper/Communication-theory/并行程序设计与算法/实验作业/lab1$ mpirun -n 8
./lab1 128 128 128
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 8
slots that were requested by the application:

  ./lab1

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process.  The number of slots available are defined by the
environment in which Open MPI processes are run:

  1. Hostfile, via "slots=N" clauses (N defaults to number of
     processor cores if not provided)
  2. The --host command line parameter, via a ":N" suffix on the
     hostname (N defaults to 1 if not provided)
  3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
  4. If none of a hostfile, the --host command line parameter, or an
     RM is present, Open MPI defaults to the number of processor cores
```

这个时候调整运行指令，使用`--oversubscribe`选项来告诉Open MPI允许进程数量超过可用的槽位数。这样就可以让Open MPI忽略槽位数的限制：

```
mpirun --oversubscribe -n 8 ./lab1 128 128 128
```

此时可以运行：



实验结果

进程数	矩阵规模				
	128	256	512	1024	2048
1	0.005689	0.06235	0.5303	7.645	131.1
2	0.005639	0.06607	0.5639	7.615	128.3
4	0.004178	0.03690	0.3321	4.543	80.01
8	0.002869	0.02578	0.2284	2.750	46.89
16	0.004536	0.03109	0.2308	2.077	27.16

实验问题

在内存受限情况下，如何进行大规模矩阵乘法计算?

可以通过以下方法：

- 使用本实验的方法，使用多核处理器或者分布式计算将大矩阵分解成较小的子矩阵，分别对这些子矩阵进行乘法操作，然后再通过点对点通信合并结果。根据内存大小确定子块的大小，仅加载小块大小的数据进入内存进行计算。
- 当矩阵包含大量的零值时，可以转换为稀疏格式存储和计算。
- 使用外部储存，用比如硬盘或网络数据库等暂时存储数据，按需加载到内存中。

如何提高大规模稀疏矩阵乘法性能?

- 使用CSR或者CSC、COO等格式对矩阵进行计算。
- 通过多线程和多核CPU并行计算，还可以使用CUDA或者OpenCL使用GPU进行并行计算。
- 使用分布式计算。
- 根据具体的硬件架构调整计算策略，比如针对特定GPU架构优化线程格局和内存访问。
- 对稀疏矩阵数据进行压缩，减少数据传输开销。