中山大学计算机学院本科生实验报告

课程名称：并行程序设计与算法

实验	基于MPI的并行矩阵乘法	专业（方向）	计算机科学与技术
学号	21307035	姓名	邓栩瀛
Email	dengxy66@mail2.sysu.edu.cn	完成日期	2024.3

1、实验目的

1.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：三个整数，每个整数的取值范围均为[128, 2048]
问题描述：随机生成的矩阵及的矩阵，并对这两个矩阵进行矩阵乘法运算，得到矩阵.
输出：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，填写表格，并分析其性能。

2.根据当前实现，在实验报告中讨论两个优化方向：a) 在内存有限的情况下，如何进行大规模矩阵乘法计算？b) 如何提高大规模稀疏矩阵乘法性能?

2、实验过程和核心代码

执行命令

```
mpic++ main.cpp -o main.out
./main.out
```

核心代码

广播输入

```
MPI_Bcast(&matrixSize, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&numProcesses, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

计算每个进程需要处理的矩阵行数

```
int rowsPerProcess = matrixSize / numProcesses;
int startRow = rank * rowsPerProcess;
int endRow = (rank == numProcesses - 1) ? matrixSize : (startRow + rowsPerProcess);
```

分配内存

```
int** matrixA = new int*[matrixSize];
int** matrixB = new int*[matrixSize];
int** result = new int*[matrixSize];
for (int i = 0; i < matrixSize; i++) {
    matrixA[i] = new int[matrixSize];
    matrixB[i] = new int[matrixSize];
    result[i] = new int[matrixSize];
}
```

生成随机矩阵

```
if (rank == 0) {
    generateRandomMatrix(matrixA, matrixSize);
    generateRandomMatrix(matrixB, matrixSize);
}
```

广播随机矩阵

```
for (int i = 0; i < matrixSize; i++) {
    MPI_Bcast(matrixA[i], matrixSize, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(matrixB[i], matrixSize, MPI_INT, 0, MPI_COMM_WORLD);
}
```

每个进程分配计算任务

```
int localRows = endRow - startRow;
int** localResult = new int*[localRows];
for (int i = 0; i < localRows; i++) {
    localResult[i] = new int[matrixSize];
}
```

并行计算矩阵乘法

```
double startTime = MPI_Wtime();
matrixMultiply(matrixA, matrixB, localResult, matrixSize, startRow, endRow);
```

收集计算结果

```
    MPI_Gather(localResult[0], localRows * matrixSize, MPI_INT, result[0], localRows * matrixSize, MPI_INT, 0,
MPI_COMM_WORLD);
```

3、实验结果

进程数	矩阵规模				
	128	256	512	1024	2048
1	0.072771	0.321411	1.34084	6.81646	time out
2	0.070127	0.276058	1.18605	5.46334	54.2342
4	0.066944	0.267516	1.0983	4.7156	35.033
8	0.064895	0.256045	1.05402	4.3564	25.5674
16	0.067656	0.255371	1.04294	4.19303	20.832

实验结果分析：

1. 随着矩阵规模的增加，运行时间也随之增加。
2. 随着进程数的增加，总体上运行时间呈现下降的趋势，这说明并行计算可以有效地减少矩阵运算的时间消耗。然而，在某些情况下，增加进程数可能会引入额外的开销，导致运行时间反而增加。
3. 对于较小的矩阵规模（如128和256），增加进程数对运行时间的影响并不明显，这是因为较小的矩阵规模可能无法充分利用并行计算的优势，并且并行化引入的通信和同步开销可能抵消了并行计算的性能提升。
4. 随着进程数的增加，性能的提升逐渐减小。从1到2个进程时，性能提升较为显著（如128规模的矩阵），但增加更多的进程时，性能提升幅度逐渐减小，这是因为并行计算的效果受限于硬件资源和并行算法的特性。

a) 在内存有限的情况下，如何进行大规模矩阵乘法计算?

从表格中可以观察到，随着矩阵的规模增加，运行时间也呈现指数级增长。在内存有限的情况下，处理大规模矩阵乘法计算可以采取以下策略：

1. 分块矩阵乘法：将大矩阵分割成小块，然后逐块进行乘法计算。这样可以减少每次计算涉及的内存使用量，降低内存压力。
2. 并行计算：利用多个处理器或计算节点并行计算矩阵乘法。根据表格中的数据，随着进程数的增加，矩阵乘法的运行时间有所下降。通过将计算任务分配给多个处理器或计算节点，可以同时处理多个部分乘法，从而提高计算效率。
3. 内存优化：优化算法和数据结构，减少内存使用量。可以使用稀疏矩阵表示方法，只存储非零元素，从而减少内存占用。此外，可以使用内存映射文件等技术，将部分数据存储在磁盘上，从而扩展可用内存空间。

b) 如何提高大规模稀疏矩阵乘法性能?

1. 稀疏矩阵存储格式：选择适合稀疏矩阵的存储格式，如压缩稀疏行或压缩稀疏列等，这些格式能够更有效地存储稀疏矩阵，减少内存使用和计算量。
2. 并行计算：利用多个处理器或计算节点并行计算稀疏矩阵乘法，将矩阵分割成多个块，每个处理器或计算节点处理一个块的乘法计算，然后合并结果。这样可以加速计算过程。
3. 算法优化：针对稀疏矩阵乘法的特点，设计高效的算法。例如，使用快速稀疏矩阵向量乘法算法，采用适合稀疏矩阵结构的优化策略，减少不必要的计算和存储操作。

4、实验感想

1. 使用MPI点对点通信方式实现并行通用矩阵乘法可以显著提高计算效率，通过将计算任务分配给多个进程，可以充分利用多核处理器或分布式计算环境的并行计算能力，加快矩阵乘法运算的速度。
2. 随着进程数量的增加，理论上可以获得更好的并行性能。然而，在实际情况中，过多的进程数量可能导致通信开销增加，从而抵消了并行计算带来的性能提升。因此，需要在进程数量选择上进行权衡和优化，找到适合特定矩阵规模和硬件环境的最佳进程数量。
3. 较大的矩阵规模通常会导致更高的计算复杂度，因此并行计算的优势更加明显。随着矩阵规模的增加，可以期望更大的性能提升。但同时，较大的矩阵规模也可能导致内存消耗增加和通信开销的增加，因此需要考虑硬件资源和通信效率来确定最佳的矩阵规模。