**深 圳 大 学 实 验 报 告**

|  |  |
| --- | --- |
| **课程名称** | **：并行计算** |
| **实验项目名称** | **：矩阵乘法的MPI并行程序** |
| **学院** | **：计算机与软件学院** |
| **专业** | **：计算机科学与技术** |
| **指导教师** | **：陆克中** |
| **报告人** | **：何泽锋** |
| **学号** | **：2022150221** |
| **实验时间** | **：2025年04月30日** |
| **实验报告提交时间** | **：2025年 05月07日** |

**教务部制**

## 一、实验目的

1. 学会编写矩阵乘法的MPI并行程序；

2. 掌握6个基本的MPI语句；

3. 对并行程序进行性能分析。

## 二、实验环境

1. 硬件环境：6×32核CPU、400GB内存的分布内存并行计算平台；

2. 软件环境：Ubuntu Linux、mpicc、mpic++（mpic++ -O3 -o a.out a.cpp）、mpiexec（mpiexec -n 16 -f machinefile /home/bxjs/a.out）

3. 远程登录：本地PowerShell中执行ssh bxjs@hpc.szu.edu.cn；

4. 传输文件：本地PowerShell中执行scp c:\a.cpp [bxjs@hpc.szu.edu.cn:/home/bxjs/](mailto:bxjs@hpc.szu.edu.cn:/home/bxjs/)或<ftp://hpc.szu.edu.cn>。

## 三、实验内容

1. 用MPI编写两个*n*阶方阵*a*和*b*的并行相乘程序，结果存放在方阵*c*中。初始时，*a*和*b*都存储在进程0中，其他进程没有数据。结束时，*c*也存储在进程0中。为了验证结果正确性，将并行计算结果和串行计算结果相比较。

2. 测试并行程序在不同进程数下的执行时间和加速比（与进程数=1时的执行时间相比）。其中，*n*固定为1000，进程数分别取1、2、4、8、16、32、64。为减少误差，每项实验进行5次，取平均值作为实验结果。

## 四、代码描述

*在下面写出完整的程序代码（文本格式，不可截图），对于其中的关键代码，以注释方式给出必要的描述。*

|  |
| --- |
| #include "mpi.h"  #include <iostream>  using namespace std;  inline double parallel\_matmul(int rank, int size, double\* a, double\* b, double\* c, int n) {  double t0 = MPI\_Wtime();  // 广播矩阵B到所有进程  MPI\_Bcast(b, n \* n, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);  // 计算各进程分配的行数  int base\_rows = n / size; // 每个进程基础行数  int remainder = n % size; // 余下的行数  int local\_rows = base\_rows + (rank < remainder ? 1 : 0); // 实际分配的行数    // 准备分发数据的参数  int\* send = new int[size] {0}; // 每个进程接收的数据量  int\* bias = new int[size] {0}; // 每个进程数据的偏移量  if (rank == 0) { // 主进程计算分发参数  int offset = 0;  for (int i = 0; i < size; i++) {  send[i] = (base\_rows + (i < remainder ? 1 : 0)) \* n; // 数据量=行数\*列数  bias[i] = offset; // 数据在矩阵中的偏移位置  offset += send[i]; // 下一个进程的偏移  }  }  // 分配本地存储空间  double\* local\_a = new double[local\_rows \* n] {0}; // 本地存储的A矩阵块  double\* local\_c = new double[local\_rows \* n] {0}; // 本地计算结果  // 分发矩阵A的数据块到各进程  MPI\_Scatterv(a, send, bias, MPI\_DOUBLE, local\_a, local\_rows \* n, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);  // 行优先计算矩阵乘法  for (int i = 0; i < local\_rows; i++) {  for (int k = 0; k < n; k++) {  double a\_ik = local\_a[i \* n + k];  double\* c\_row = &local\_c[i \* n];  double\* b\_row = &b[k \* n];  for (int j = 0; j < n; j++) {  c\_row[j] += a\_ik \* b\_row[j];  }  }  }  // 收集各进程计算结果到主进程  MPI\_Gatherv(local\_c, local\_rows \* n, MPI\_DOUBLE, c, send, bias, MPI\_DOUBLE,0, MPI\_COMM\_WORLD);  delete[] local\_a;  delete[] local\_c;  delete[] send;  delete[] bias;  double t1 = MPI\_Wtime();  return t1 - t0;  }  int main(int argc, char\* argv[]) {  const int n = 1000;  double\* a = new double[n \* n];  double\* b = new double[n \* n];  double\* c = new double[n \* n];  int rank, size;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  if (rank == 0)  {  srand(0);  for (int i = 0; i < n; i++)  for (int j = 0; j < n; j++)  {  a[i \* n + j] = rand() \* 1.0 / RAND\_MAX;  b[i \* n + j] = rand() \* 1.0 / RAND\_MAX;  }  }  double total\_time = 0;  int num\_runs = 5; // 多次运行取均值  for (int run = 0; run < num\_runs; run++) {  MPI\_Barrier(MPI\_COMM\_WORLD); // 同步所有进程  double t = parallel\_matmul(rank, size, a, b, c, n);    if (rank == 0) {  total\_time += t;  cout << t<< endl;  }  }  // 主进程输出平均耗时  if (rank == 0&& num\_runs != 1) {  double avg\_time = total\_time / num\_runs;  cout << avg\_time << endl;  }  // 主进程验证结果正确性  if (rank == 0)  {  for (int i = 0; i < n; i++)  for (int j = 0; j < n; j++)  {  double s = 0;  for (int k = 0; k < n; k++)  s += a[i \* n + k] \* b[k \* n + j];  if (s != c[i \* n + j]) {  cout << "error" << endl;  break;  }  }  cout << "PASS" << endl;  }  if (rank == 0) {  delete[] a;  delete[] b;  delete[] c;  }  else {  delete[] b;  }  MPI\_Finalize();  return 0;  } |

## 五、实验结果和分析

*实验结果以及对实验结果的比较分析和综合概括。*

表1 并行程序在不同进程数下的执行时间（秒）和加速比（*n*=1000）（记录广播时间）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 进程数  执行时间 | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 第1次 | 0.47179 | 0.37322 | 0.35530 | 0.36780 | 0.44881 | 0.68928 | 1.05395 |
| 第2次 | 0.46005 | 0.36356 | 0.34717 | 0.34444 | 0.41493 | 0.66661 | 1.31411 |
| 第3次 | 0.45941 | 0.35455 | 0.34808 | 0.35443 | 0.41346 | 0.67781 | 1.08523 |
| 第4次 | 0.46658 | 0.35433 | 0.35937 | 0.35432 | 0.41204 | 0.67269 | 1.00657 |
| 第5次 | 0.46297 | 0.35425 | 0.35314 | 0.35354 | 0.43106 | 0.62994 | 0.98590 |
| 平均值 | 0.46416 | 0.35998 | 0.35261 | 0.35491 | 0.42406 | 0.66727 | 1.08915 |
| 加速比 | 1.00000 | 1.28941 | 1.31636 | 1.30783 | 1.09457 | 0.69562 | 0.42617 |

实验在计算不同进程数的情况下的数据时，首先记录了包含矩阵B广播的运行时间，可以发现随着进程数量的增加（大于实际的机器数量时），运行时间反而会加长，分析知道这是因为广播矩阵B到每个进程需要一定时间，因此不能很好的反映矩阵乘法上的加速，得到的图如下所示，可以看到，时间先下降后上升，加速比呈相反趋势，先下降后下降。

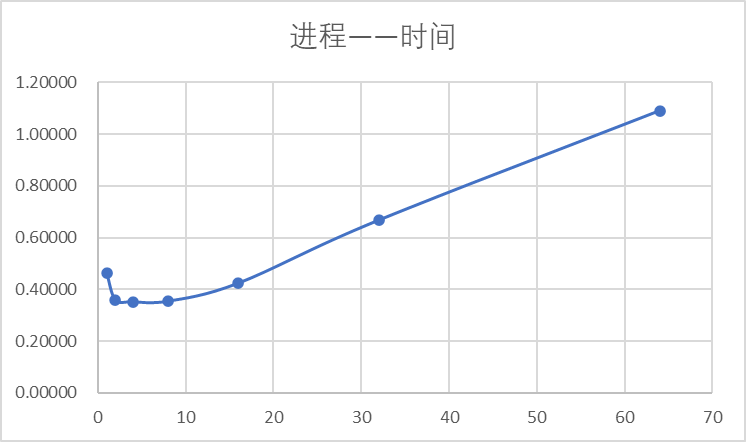
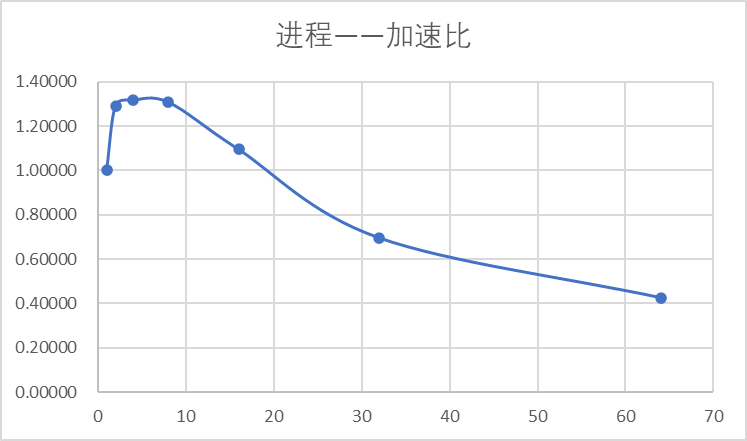
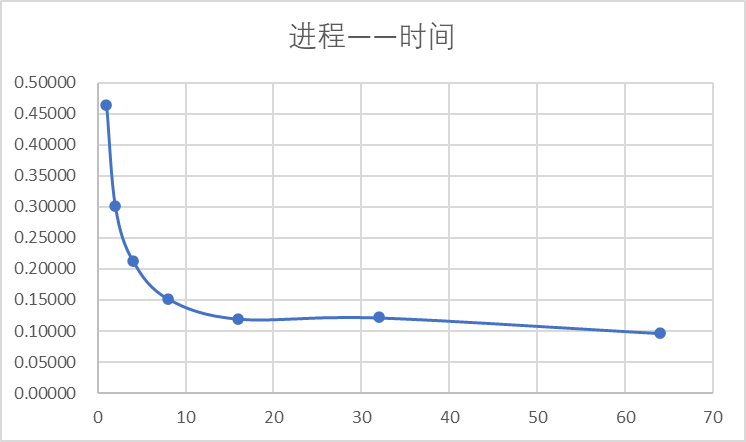
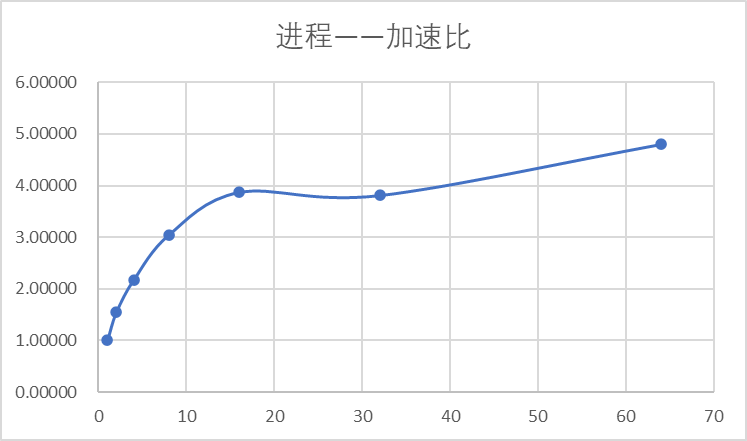
 

表2 并行程序在不同进程数下的执行时间（秒）和加速比（*n*=1000）（忽略广播时间）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 进程数  执行时间 | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 第1次 | 0.47232 | 0.30463 | 0.21649 | 0.15108 | 0.11637 | 0.13161 | 0.09686 |
| 第2次 | 0.46048 | 0.29936 | 0.21217 | 0.15554 | 0.11836 | 0.13503 | 0.09577 |
| 第3次 | 0.46017 | 0.29955 | 0.21220 | 0.15158 | 0.12135 | 0.11232 | 0.09642 |
| 第4次 | 0.46660 | 0.29915 | 0.21305 | 0.15147 | 0.12215 | 0.11526 | 0.09791 |
| 第5次 | 0.46217 | 0.30090 | 0.21221 | 0.15133 | 0.12108 | 0.11456 | 0.09644 |
| 平均值 | 0.46435 | 0.30072 | 0.21322 | 0.15220 | 0.11986 | 0.12175 | 0.09668 |
| 加速比 | 1.00000 | 1.54414 | 2.17777 | 3.05089 | 3.87409 | 3.81388 | 4.80304 |

为了得到更真实的矩阵计算时间变化，此处忽略广播的时间，可以看到，此时随着进程数量的增加，时间是逐渐下降的，初始的时候下降快，进程数较大时下降较慢；而加速比也是逐渐上升的

## 六、实验结论

*实验过程中遇到的问题及解决办法，运用了哪些技术方法以提高实验性能，从该实验得到的客观结论，等等。*

本次实验通过MPI实现了矩阵乘法的并行计算，并分析了并行程序在不同进程数下的性能。实验发现，矩阵B的广播操作在进程数较多时成为性能瓶颈。忽略广播时间后，加速比曲线更接近理想状态。通过数据分析可知，随着进程数的增加，程序所需时间下降的速率逐渐变缓（加速比上升的速度也逐渐变缓），说明并行技术并不能无限制的优化程序，只能在有限范围内加速。因此本次实验也尝试使用了其他优化方式，通过使用行优先的循环遍历方式，增加cache的命中率，进而减少内存调用的频率，加速了矩阵计算。