MPI 编程实验
——以高斯消去为例

杜忱莹 周辰霏
2021 年 5 月
麦隽韵
2022 年 5 月
李君龙
2023 年 5 月
丁延峰
2024 年 5 月

目录

1 实验介绍

1.1 实验选题

1. 默认选题：高斯消去法 (LU 分解)。
2. 鼓励自选与期末研究报告结合的题目，选题建议和要求同前次作业。

1.2 实验要求

1. 基本要求（最高获得 80% 分数）：ARM 平台（华为服务器）或 x86 平台（自行安装）上普通高斯消去计算的基础 MPI 并行化实验：
   - 设计实现适合的任务分配（数据划分）算法，分析其性能。
   - 在 ARM 平台或 x86 平台上编程实现、进行实验，测试不同问题规模、不同节点数/线程数下的算法性能（串行和并行对比），讨论一些基本的算法/编程策略对性能的影响。
2. * 进阶要求（最高可获得剩余 20% 分数）：进一步
   - 探讨与多线程（Pthread 或 OpenMP）和 SIMD 的结合。
   - 探讨特殊的高斯消去计算的 MPI 并行化。
   - 不同平台（如 x86）上并行化实验。
   - MPI 并行化的不同算法策略（如块划分、循环划分等不同任务划分方法，流水线算法等）及其复杂性分析（如并行时间、通信开销、加速比、效率）。
   - 不同 MPI 编程方法、通信机制的对比，如阻塞通信 vs. 非阻塞通信，双边通信 vs. 单边通信、MPI 自身的多线程支持等。
   - profiling 及体系结构相关优化（如 cache 优化）等。
3. 自主选题视难度和工作量与默认题目对等评分。
4. 撰写研究报告（问题描述（特别是对自主选题，首先简要描述期末研究报告的大问题，然后具体描述本次 MPI 编程实验涉及的子问题）、MPI+Multi-Threads+SIMD 算法设计（最好有复杂性分析）与实现、实验及结果分析），符合科技论文写作规范，附 Git 项目链接。

2 多进程与多线程介绍

本次实验涉及到的 MPI 编程是多进程编程，什么是进程？本节介绍了线程和进程的区别于联系。首先明确：OpenMP 和 Pthread 均为多线程编程。

2.1 进程与线程

进程 (process) 是操作系统进行资源分配的最小单元，**线程 (thread)** 是操作系统进行运算调度的最小单元。

通常情况下，我们在运行程序时，比如在 Linux 下通过 *./test* 运行一个可执行文件，那么我们就相当于创建了一个进程，操作系统会为这个进程分配 ID 和堆栈空间等资源。如果这个可执行程序是个多线程程序（比如由 OpenMP 或 Pthread 编写的 cpp 文件编译而来），那么这个 *./test* 进程在执行到某个特定位置时会创建多个线程继续执行。所以，**进程可以包含多个线程，但每个线程只能属于一个进程**。

如图2.1所示，每个进程有独立的地址空间（外部框表示），同一进程内不同线程（曲线）共享该进程的内存。但是不同进程不共享地址空间，如果一个进程要访问另一个进程的数据就只能通过特定的函数进行通信。

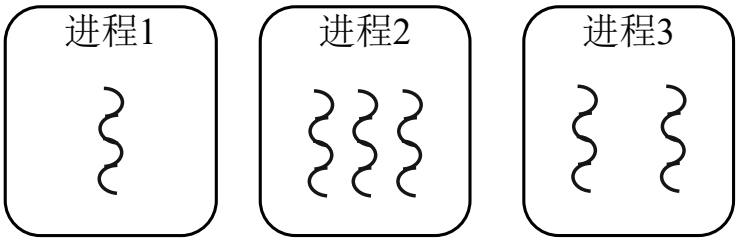

图 2.1: 进程与线程概念

2.2 MPI 多进程

区别于普通的可执行程序只创建一个进程，通过 $mpiexec -n\ num\ ./test$ 运行的程序会根据 $-n$ 选项后的参数 num，创建 num 个进程运行 $test$ 可执行程序。

课程中提到了在开始编写 MPI 部分代码时调用 MPI_Init 完成 MPI 初始化，在 MPI 部分结束时调用 MPI_Finalize 完成清理工作。这两个函

数与多线程中创建和销毁线程有本质的区别：程序并非只在两个函数调用之间的部分并行执行，而是整体并行执行，在两个函数之间的部分完成数据划分运算、通信等工作。以下方程序为例，代码在两个函数调用的外部定义了 $size$ 变量和 $rank$ 变量，同时输出 $HelloWorld!$；在两个函数之间的部分为 $size$ 和 $rank$ 赋值并输出。编译并通过 $mpiexec\ -n\ num\ ./test$ 运行后，每个进程均会输出一次 $HelloWorld!$，共输出 num 次，这也验证并行并非只发生在 MPI_Init 和 MPI_Finalize 之间的代码。

```
#include"mpi.h"
#include<iostream>

int main(int argc, char* argv[]) {
    int size,rank;
    std::cout<<"Hello world!"<<std::endl;
    MPI_Init(&argc, &argv);
    size = MPI::COMM_WORLD.Get_size();
    rank = MPI::COMM_WORLD.Get_rank();
    std::cout<<"MPI size:"<<size<<" ,rank:"<<rank<<std::endl;
    MPI_Finalize();
    return 0;
}
```

3 实验设计指导

3.1 实验总体思路

以高斯消去法为例：

1. 首先初始化生成矩阵元素值，按照之前作业给出的伪代码实现高斯消去法串行算法。

```
procedure LU (A)
begin
 for k := 1 to n do
   for j := k+1 to n do
     A[k, j] := A[k, j]/A[k, k];
   endfor;
   A[k, k] := 1.0;
   for i := k + 1 to n do
     for j := k + 1 to n do
       A[i, j] := A[i, j] − A[i, k] × A[k, j];
       endfor;
   A[i, k] := 0;
```

```
13      endfor;
14    endfor;
15  end LU
```

2. 使用课堂所学 MPI 编程设计实现合适的任务分配（数据划分）算法，按照不同任务划分方式（如块划分、循环划分等），分别设计并实现 MPI 算法。并考虑将其与 Pthread 算法以及 SIMD 算法结合。对各算法进行复杂度分析（基本的运行时间、加速比的分析以及更深入的伸缩性分析），思考能否继续改进。

以 MPI 按一维（行）块划分消去并行处理为例，假设设置 m 个 MPI 进程、问题规模为 n，则给每一个进程分配 n/m 行的数据，对于第 i 个进程，其分配的范围为 $[i*(n-n\%m)/m, i*(n-n\%m)/m+(n-n\%m)/m-1]$，而最后一个进程分配 $[(m-1)*(n-n\%m)/m, n-1]$ 行。注意，这种简单策略是将余数部分（$n\%m$ 行）都分配给了最后一个进程，大家可思考稍复杂些但负载更为均衡的分配策略——将余数部分均匀分配给前 $n\%m$ 个进程，每个进程一行。初始化矩阵等工作由 0 号进程实现，然后将分配的各行发送给各进程。在第 k 轮消去步骤，负责第 k 行的进程进行除法运算，将除法结果一对多广播给编号更大的进程，然后这些进程进行消去运算。消除过程完成后，可将结果传回 0 号进程进行回代，也可由所有节点进行并行回代。

一维块循环划分略微复杂，主要是以更小任务粒度循环给各进程分配任务行号的计算会更复杂些，大家自行推导。一维列（循环）块划分与一维行划分略有不同，在除法阶段，需要持有对角线上元素的进程将其广播给其他进程，然后所有进程对自己所负责的列元素进行除法计算；接下来无需广播除法结果，因为需要除法结果的后续行都由同一个进程负责，直接在本地进行消去计算即可。这里就有一些可优化之处，大家可以考虑采用非阻塞通信、单边通信等手段降低对角线元素广播带来的进程空闲等待时间。

二维划分就更为复杂一些，大家需要更小心地计算每个进程负责的行号、列号，安排好通信，显然，采取二维划分后，既需要列方向广播对角线元素，也需要行方向广播除法结果。

流水线算法和普通的块划分的区别在于一个进程负责行的除法运算完成之后，并不是将除法结果一对多广播给所有后续进程，而是（点对点）转发给下一个进程；当一个进程接收到前一个进程转发过来的除法结果时，首先将其继续转发给下一个进程，然后再对自己所负责的行进行消去操作；当一个进程对第 k 行完成了第 $k-1$ 个消去步骤的消去运算之后，它即可对

Algorithm 1: MPI 版本的普通高斯消元

Data: 系数矩阵 A[n,n], 进程号 myid, 负责的起始行 r1, 负责的终止行 r2, 进程总数 num

Result: 上三角矩阵 A[n,n]

```
for k = 0 to n-1 do
    if r1 <= k <= r2 then
        for j = k + 1; j < n; j++ do
            A[k,j]=A[k,j]/A[k,k];
        A[k,k] ← 1.0;
        for j = 0; j < num; j++ do
            MPI_Send(&A[k,0], n, MPI_FLOAT, j, ...);
    else
        MPI_Recv(&A[k,0], n, MPI_FLOAT, j, ...);
    for i ← r1 to r2 do
        for j = k + 1; j < n; j++ do
            A[i,j] ← A[i,j]-A[k,j]*A[i,k];
        A[i,k] ← 0;
    k++;
    ...
```

第 k 行进行第 k 个消去步骤的除法操作，然后将除法结果进行转发，如此重复下去，直至第 $n-1$ 个消去步骤完成。

3. 实验方面，改变矩阵的大小、进程数等参数，观测各算法运行时间的变化，对结果进行性能分析。测试相同并发度 (总线程数) 下不同节点数与每节点线程数的组合，并借助 Vtune profiling 等工具分析算法过程中的同步开销和空闲等待等。

3.2 MPI 的 C++ 编程

1. **头文件**

   所有进行 MPI 调用的程序单元必须包括 “mpi.h” 头文件。该文件定

义了一些 MPI 常量，并提供了 MPI 函数原型。

```
#include "mpi.h"
```

2. MPI 预定义数据类型

```
MPI_CHAR
MPI_SHORT
MPI_INT
MPI_LONG
MPI_UNSIGNED_CHAR
MPI_FLOAT
MPI_DOUBLE
...
```

3. 常用的 MPI 函数

```
MPI_Comm_size//报告进程数
int MPI_Comm_size(MPI_Comm comm, int *size);

MPI_Comm_rank//报告识别调用进程的 rank，值从 0 size-1
int MPI_Comm_rank(MPI_Comm comm, int *rank);

MPI_Init//令 MPI 进行必要的初始化工作
int MPI_Init(
    int*  argc_p /* 输入/输出参数 */,
    char *** argv_p /* 输入/输出参数 */);

MPI_Finalize//告诉 MPI 程序已结束，进行清理工作
int MPI_Finalize(void);

MPI_Send//基本（阻塞）发，向一个进程发送数据
int MPI_Send(
    void* buf  /* 存储数据的缓冲区地址 */,
    int count  /* 发送的数据量 */,
    MPI_Datatype datatype /* 数据类型 */,
    int dest  /* 目的进程编号 */,
    int tag  /* 标识向同一个目的进程发送的不同数据 */,
    MPI_Comm /* MPI集群的通信域标识 */);

MPI_Recv//基本（阻塞）接收，从一个进程接收数据
int MPI_Recv(
    void* buf  /* 存储数据的缓冲区地址 */,
    int count  /* 接收的数据量 */,
    MPI_Datatype datatype /* 数据类型 */,
    int source  /* 源头进程编号 */,
```

```
30      int tag  /* 标识从同一个源头进程接收的不同数据 */,
31      MPI_Comm /* MPI集群的通信域标识 */,
32      MPI_Status *status /* 可以记录更多额外的信息 */);
```

更多更详细的内容请参考 MPI 的讲义。

3.3 作业注意要点及建议

1. 矩阵数值初始化问题

根据有些同学们反映，自己初始化矩阵在计算过程中会出现 inf 或 nan 的问题。这是由于精度问题以及非满秩矩阵造成的。inf 或 nan 的情况无疑会影响结果正确性的判断，也会在并行计算性能上造成一定影响，而随机生成数据的方式很明显无法保证能避免该问题尤其在规模巨大的情况下。个人建议初始化矩阵时可以首先初始化一个上三角矩阵，然后随机的抽取若干行去将它们相加减然后执行若干次，由于这些都是内部的线性组合，这样的初始数据可以保证进行高斯消去时矩阵不会有 inf 和 nan。

2. 不同任务划分策略

前面介绍了 MPI 任务分配（数据划分）策略，在此基础上，可结合多线程以及 SIMD 并行进行任务划分。例如，对于一维行划分，可看作是将第二层循环拆分，分配给不同进程；这样，继续进行多线程并行，即可继续对第二层循环进行划分，即，将进程负责的行划分给内部的多个线程，也可以对最内层循环进行划分，即，将进程负责的所有行的不同列分配给不同线程；而再继续结合 SIMD 并行化，则只能对最内层循环进行向量化。MPI 列划分、二维划分下与多线程和 SIMD 的结合类似，大家自己思考。

3. 计算误差与程序正确性

有关问题规模和并行计算由于重排了指令执行顺序和计算机浮点数所导致误差问题说明参考之前实验。

由于 n 较小的时候测出的计算时间也较小, 误差较大, 所以建议采取多次测量取均值的方法确定较合理的性能测试结果, 同时保证几种算法重复次数一致, 减少误差。而且考虑到实验平台等因素，计时测试工具可以考虑使用 MPI 计时。

4 鲲鹏服务器作业提交说明

4.1 pbs 脚本配置

提交任务前需配置脚本，例如运行 4 个进程 mpi 程序可以编写 mpi.sh 脚本内容如下：

```
# mpi.sh
# !/bin/sh
# PBS -N mpi      # 任务程序为mpi
# PBS -l nodes=4 # 需要四个节点来计算

pssh -h $PBS_NODEFILE mkdir -p /home/sTest 1>&2 # 在分配到的4个计算节点上创建对应路径
scp master:/home/sTest/mpi /home/sTest       # mpi为编译之后的可执行程序
pscp -h $PBS_NODEFILE /home/sTest/mpi /home/sTest 1>&2
# 获取master节点中的mpi可执行程序，并分发到每个计算节点
mpiexec -np 4 -machinefile $PBS_NODEFILE /home/sTest/mpi
# 在4个计算节点上运行可执行程序mpi
```

脚本通过 $PBS\ -l\ nodes = 4$ 指定分配的节点数，部分同学可能在申请某些节点时出问题，此时可以手动指定分配的节点，多个节点之间用 + 号连接，例如 $PBS\ \ -l\ nodes = master\_vir1 + master\_vir2$。节点列表通过 $pbsnodes\ \ -l\ free$ 指令查看。

注意，对单节点非 MPI 实验，每个任务只能使用 1 个节点（核心）、运行一个进程，节点数设置为 1，无须使用 mpiexec，直接执行程序即可。对于非 MPI 实验，可以编写 test.sh 脚本如下：

```
#test.sh
#!/bin/sh
#PBS -N test      # 任务程序为test
#PBS -l nodes=1 # 需要一个节点来计算

pssh -h $PBS_NODEFILE mkdir -p /home/sTest 1>&2
scp master:/home/sTest/test /home/sTest
pscp -h $PBS_NODEFILE /home/sTest/test /home/sTest 1>&2
/home/sTest/test
```

注意事项: 如果进行特殊高斯消去及其他需要从本地文件读入数据的实验，还需要通过 pbs 脚本将想读的文件和可执行文件一起发送至计算节点，读文件时使用绝对路径，如下所示。

```
# index.sh
# !/bin/sh
#PBS -N index
#PBS -l nodes=1

pssh -h $PBS_NODEFILE mkdir -p /home/sTest/sTest 1>&2
scp master:/home/sTest/sTest/index /home/sTest/sTest
pscp -h $PBS_NODEFILE /home/sTest/sTest/index /home/sTest/sTest 1>&2
scp master:/home/sTest/sTest/ExpQuery.txt /home/sTest/sTest
pscp -h $PBS_NODEFILE /home/sTest/sTest/ExpQuery.txt /home/sTest/sTest 1>&2
/home/sTest/sTest/index
```

图 4.2: 作业提交

4.2 作业提交

我们利用鲲鹏服务器搭建虚拟机集群供同学们进行 MPI 编程实验，包括一个 master 节点和 22 个计算节点，每个节点 8 核，12GB 内存。登录管理节点 IP 地址 10.137.144.91 (需要链接校园网)，端口号为 9001，登录用户名和密码均为 s+ 学号，例学号为 2211111，则用户名和密码均为 s2211111。

登录管理节点后，将程序编译成可执行文件（mpicc/mpic++，与自行搭建的 OpenMPI/MPICH 系统中类似），在可执行文件所在目录下通过提交 pbs 脚本，向系统提交计算任务（**禁止在管理节点直接执行程序**），具体步骤如下：

```
1 mpic++ mpi.cpp -o test // 编译源码为可执行文件
2 qsub test.sh      // 提交pbs脚本给管理节点，由作业系统分配计算节点给可执行文件进行计算
```

之后管理节点会根据脚本内容将作业任务分配给计算节点，可以通过 qstat 查看当前队列中正在执行的作业状态：

```
1 Q—正在排队 R—正在运行 C—运行完毕
```

作业运行完毕后，当前目录下会多出两个文件：“程序名.o 作业编号”，“程序名.e 作业编号”，分别代表标准输出和错误输出。

可以看到如图4.3所示结果：

```
[sTest@master mpi]$ qsub mpi_test.sh
9269.master
[sTest@master mpi]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
9269.master                mpi_test.sh      sTest            00:00:00 C dque
```

图 4.3: 作业提交

提交后得到作业 9269.master。开始状态为 R，表示正在运行，之后状态为 C，表示运行结束。在程序运行结束后，在当前目录下出现两个文件：test.o9269、test.e9269。分别表示 9269 作业的标准输出和错误输出。

PBS 脚本编写更多参数含义可参考，pssh 命令的使用参考。