MPI 编程实验
——以高斯消去为例

杜忱莹 周辰霏

May 2021

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. 对选定题目设计实现 MPI 算法。
   - 设计实现适合的任务分配 (数据划分) 算法，分析其性能。
   - 与 pthread/OpenMP 以及 SIMD(SSE/AVX/Neon) 算法结合。
2. 提交研究报告（问题描述、MPI+Multi-Threads+SIMD 算法设计与实现、实验及结果分析）和源码（只将程序文件和工程文件提交，不要将编译出的目标文件和可执行文件也打包提交）。

1.3 思考

考虑更多算法设计、MPI 编程、性能分析/测试的问题，以高斯消去为例：

1. **算法方面的探讨：** 不同的任务分配策略 (块划分、循环块划分等)、流水线算法 (学习补充的讲义音频讲解) 等，相应的复杂性分析 (并行时间、通信开销、加速比、效率等)。
2. **编程上的探讨：** 不同的通信机制对比——阻塞通信 vs. 非阻塞通信 (组通信)、双边通信 vs. 单边通信 (RMA)、MPI 自身多线程支持等。
3. **实验设计**、**结果分析**、**MPI profiling**，与算法复杂性分析的对照、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、arm 平台上的实验、与 x86 平台的对比等等。

2 实验设计指导

2.1 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。

```
1  procedure LU (A)
2  begin
3   for k := 1 to n do
4     for j := k+1 to n do
5       A[k, j] := A[k, j]/A[k, k];
6     endfor;
7     A[k, k] := 1.0;
8     for i := k + 1 to n do
9       for j := k + 1 to n do
10        A[i, j] := A[i, j] − A[i, k] × A[k, j ];
11        endfor;
12    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 m 个 MPI 进程，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，主要是以更小任务力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（参见 QQ 群上传的补充材料）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、线程数等参数，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。

2.2 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。

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

2. **MPI 预定义数据类型**

```
1 MPI_CHAR
2 MPI_SHORT
3 MPI_INT
4 MPI_LONG
5 MPI_UNSIGNED_CHAR
6 MPI_FLOAT
7 MPI_DOUBLE
8 ...
```

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);
```

更多更详细的内容请参考 MPI 的讲义。

2.3 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。

3 金山云作业提交说明

3.1 作业提交

我们还将基于金山云搭建虚拟机集群供同学们进行 MPI 编程实验，其架构如图3.1所示。虚拟机集群包括 1 个 master 节点和 32 个计算节点，每

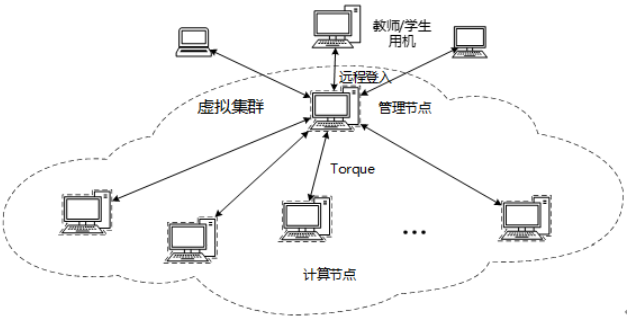

图 3.1: 虚拟机集群

个计算节点 2 核，共 64 核可供计算。节点的 CPU 支持 avx512 指令集，因此大家在结合 SIMD 的实验中，可采用 avx512 尝试更高并发度的 SIMD 并行化。集群采用管理软件 Torque 管理作业提交计算，这是基于 pbs 的一个开源版本的分布式作业提交系统。

学生登录管理节点的 IP 地址为 120.92.20.82；用户名为 s+ 学号；密码为问卷收集设定初始密码。可以使用 MobaXterm、PuTTY 等工具远程连接服务器，将源程序和数据上传到管理节点并通过集群进行计算。

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提交计算任务的步骤如下（**禁止在管理节点直接执行程序**）：

```
1 mpic++ mpi.cpp //编译源码为可执行文件
2 qsub test.sh     //提交pbs脚本给管理节点
```

pbs 执行脚本内容含义如下：

```
#test.sh
#!/bin/sh
#PBS -N test    # 该任务名称为test
#PBS -l nodes=4 # 该任务需要四个节点计算（尽量不要使用过多节点以免影响其他同学的任务运行）
pssh -h $PBS_NODEFILE mkdir -p /home/s1800001/test 1>&2 #在分配节点上创建执行路径
pscp.pssh -h $PBS_NODEFILE /home/s1800001/test/mpi /home/s1800001/test 1>&2 #将管理节点上的可执行文件分发到各计算节点
mpiexec -np 4 -machinefile $PBS_NODEFILE /home/s1800001/test/mpi #在4个计算节点上运行mpi程序cpi
```

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：

```
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3.2所示结果：

```
[s1712991@master test1]$ qsub test.sh
20.master
[s1712991@master test1]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
19.master                  mytask           s1712991        00:00:00 R master
20.master                  mytask           s1712991               0 R master
```

图 3.2: 作业提交

提交后得到作业 20.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.o20、test.e20。分别表示 20 作业的标准输出和错误输出。打开 test.o20 可以看到程序的标准输出 (图3.3)。

```
Process 0 of 4 is on node4
Process 2 of 4 is on node2
Process 1 of 4 is on node3
pi is approximately 3.1415926544231239, Error is 0.0000000008333307
wall clock time = 0.001947
Process 3 of 4 is on node1
```

图 3.3: cpi.o129

打开 test.e20，可以看到程序的错误输出如图3.4。这些是节点间文件同步操作的输出信息，和 cpi 程序无关，是 test.sh 脚本中 pssh 的输出。

PBS 脚本编写更多参数含义可参考，pssh 命令的使用参考

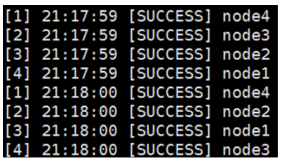
[1] 21:17:59 [SUCCESS] node4
[2] 21:17:59 [SUCCESS] node3
[3] 21:17:59 [SUCCESS] node2
[4] 21:17:59 [SUCCESS] node1
[1] 21:18:00 [SUCCESS] node4
[2] 21:18:00 [SUCCESS] node2
[3] 21:18:00 [SUCCESS] node1
[4] 21:18:00 [SUCCESS] node3

图 3.4: cpi.e129