

实验报告

实验名称	实验四 集群结构体验——MPI 编程		
实验教室	丹青 911	实验日期	2022 年 11 月 15 日
学 号	2019210173	姓 名	刘思远
专业班级	奥林学院计算机科学与技术 04 班		
指导教师	卢洋		

东北林业大学
信息与计算机科学技术实验中心

一、 实验目的

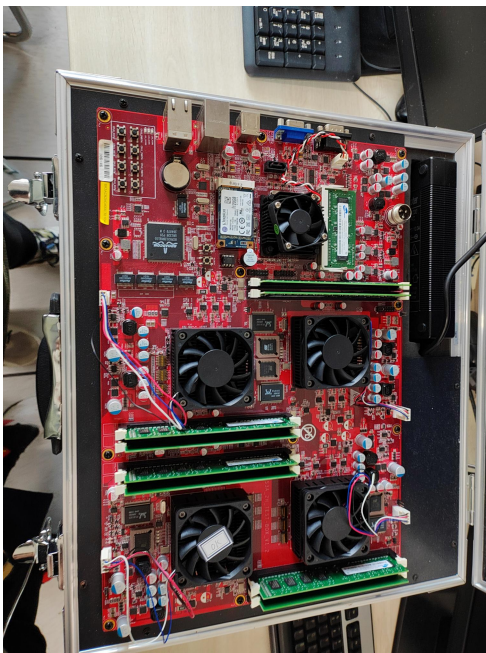
1. 本实验用 MPI 完成 Cannon 算法的并行计算。在 4 路 4 核龙芯的配置环境下，调整 MPI 进程数量（1~16），计算加速比，分析通信在 MPI 程序中对加速比的影响。注意 MPI 中消息传递的使用，并对比实验 10 中与 SMP 结构中使用共享内存的差异。

二、 实验环境

多路处理计算器

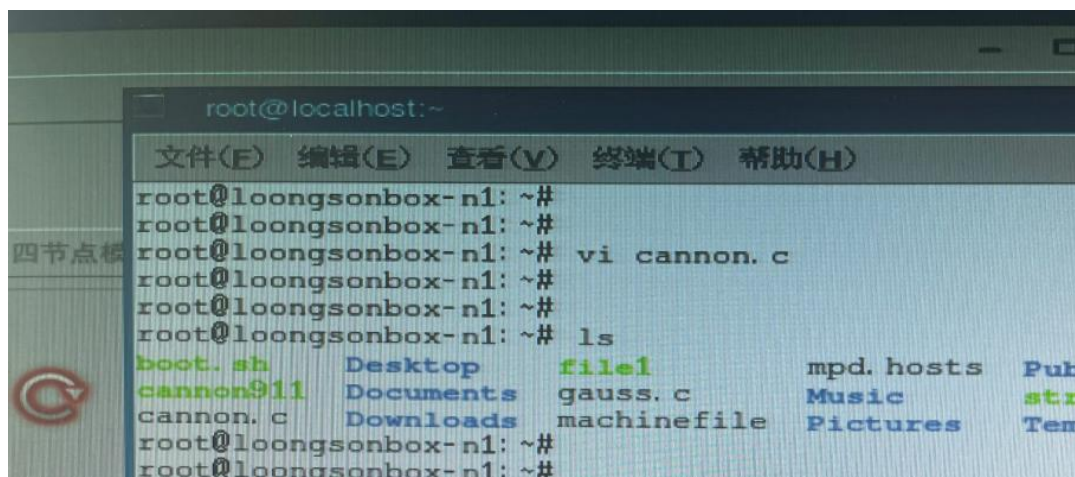
三、 实验内容及结果

1. 首先，将外接设备接好计算器，并启动计算器

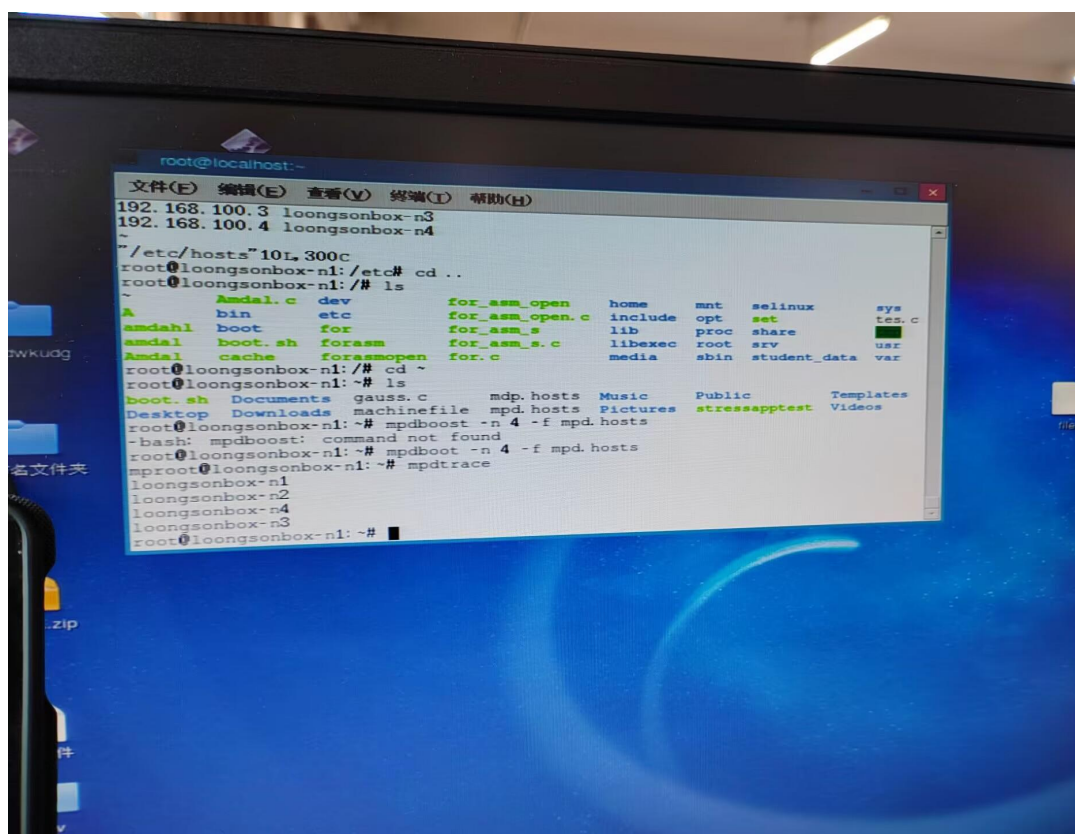


2. 按照实验要求，进行在软件启动多路处理系统软件主界面上

点单节点模式 (Single Node)，并点 4 个计算节点开机按钮使 4 个 3A 计算节点开机启动运行，启动后均运行于 linux 操作系统中。并成功建立和启动分布式系统
开启四个节点



```
root@localhost:~  
文件(E) 编辑(E) 查看(V) 终端(T) 帮助(H)  
root@loongsonbox-n1: ~#  
root@loongsonbox-n1: ~#  
root@loongsonbox-n1: ~# vi cannon.c  
root@loongsonbox-n1: ~#  
root@loongsonbox-n1: ~#  
root@loongsonbox-n1: ~# ls  
boot.sh Desktop file1 mpd.hosts Pub  
cannon911 Documents gauss.c Music str  
cannon.c Downloads machinefile Pictures Tem  
root@loongsonbox-n1: ~#  
root@loongsonbox-n1: ~#
```



```
root@localhost:~  
文件(E) 编辑(E) 查看(V) 终端(T) 帮助(H)  
192.168.100.3 loongsonbox-n3  
192.168.100.4 loongsonbox-n4  
~/etc/hosts 101.300c  
root@loongsonbox-n1: /etc# cd ..  
root@loongsonbox-n1: ~# ls  
A amdahl.c dev for_asm_open home mnt selinux sys  
amdahl bin etc for_asm_open.c include opt set tes.c  
amdahl boot.sh forasm for_asm_s lib proc share  
Amdahl cache forasmopen for.c media sbin student_data var  
root@loongsonbox-n1: ~# cd ~  
root@loongsonbox-n1: ~# ls  
boot.sh Desktop Downloads gauss.c mpd.hosts Music Public Templates  
root@loongsonbox-n1: ~# mpdboost -n 4 -f mpd.hosts  
-bash: mpdboost: command not found  
root@loongsonbox-n1: ~# mpdboot -n 4 -f mpd.hosts  
mproot@loongsonbox-n1: ~# mpdtrace  
loongsonbox-n1  
loongsonbox-n2  
loongsonbox-n4  
loongsonbox-n3  
root@loongsonbox-n1: ~#
```

3. 用 MPI 完成 Cannon 算法的并行计算

主控服务器机上使用: scp cannon.c root@192.168.100.1

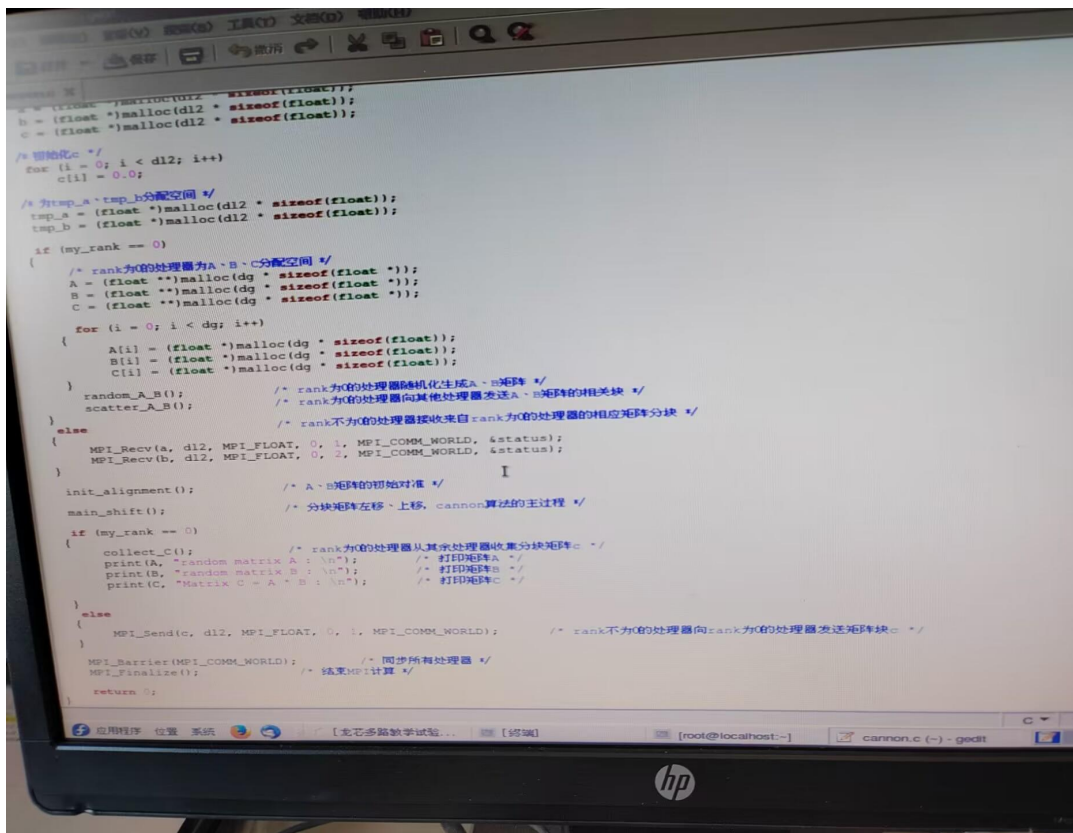
节点 1 上使用编译命令为：

```
mpicc -lm -o cannon911 cannon.c
```

运行程序命令（节点 1 上运行）：

```
mpirun -np 16 ./cannon911 4
```

编写好的程序



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define N 1024
#define D12 128
#define D2 64
#define D4 32

int main(int argc, char **argv)
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* 初始化 */
    for (i = 0; i < D12; i++)
        c[i] = 0.0;

    /* 为tmp_a, tmp_b分配空间 */
    tmp_a = (float *)malloc(D12 * sizeof(float));
    tmp_b = (float *)malloc(D12 * sizeof(float));

    if (my_rank == 0)
    {
        /* rank为0的处理单元为A、B、C分配空间 */
        A = (float **)malloc(Dg * sizeof(float *));
        B = (float **)malloc(Dg * sizeof(float *));
        C = (float **)malloc(Dg * sizeof(float *));

        for (i = 0; i < Dg; i++)
        {
            A[i] = (float *)malloc(Dg * sizeof(float));
            B[i] = (float *)malloc(Dg * sizeof(float));
            C[i] = (float *)malloc(Dg * sizeof(float));
        }

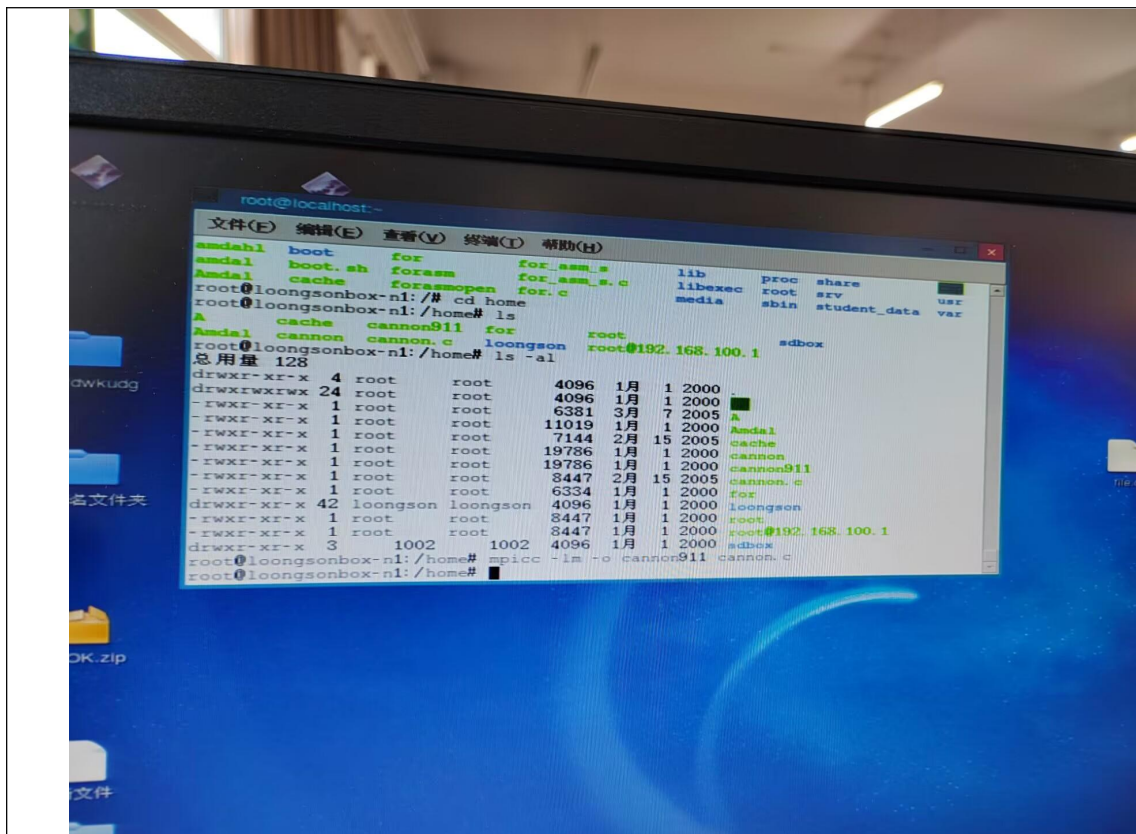
        random_A_B(); /* rank为0的处理单元随机生成A、B矩阵 */
        scatter_A_B(); /* rank为0的处理单元向其他处理单元发送A、B矩阵的相关块 */
    }
    else /* rank不为0的处理单元接收来自rank为0的处理单元的相应矩阵分块 */
    {
        MPI_Recv(a, D12, MPI_FLOAT, 0, 1, MPI_COMM_WORLD, &status);
        MPI_Recv(b, D12, MPI_FLOAT, 0, 2, MPI_COMM_WORLD, &status);
    }

    init_alignment(); /* A、B矩阵的初始对准 */
    main_shift(); /* 分块矩阵左移、上移, cannon算法的主过程 */

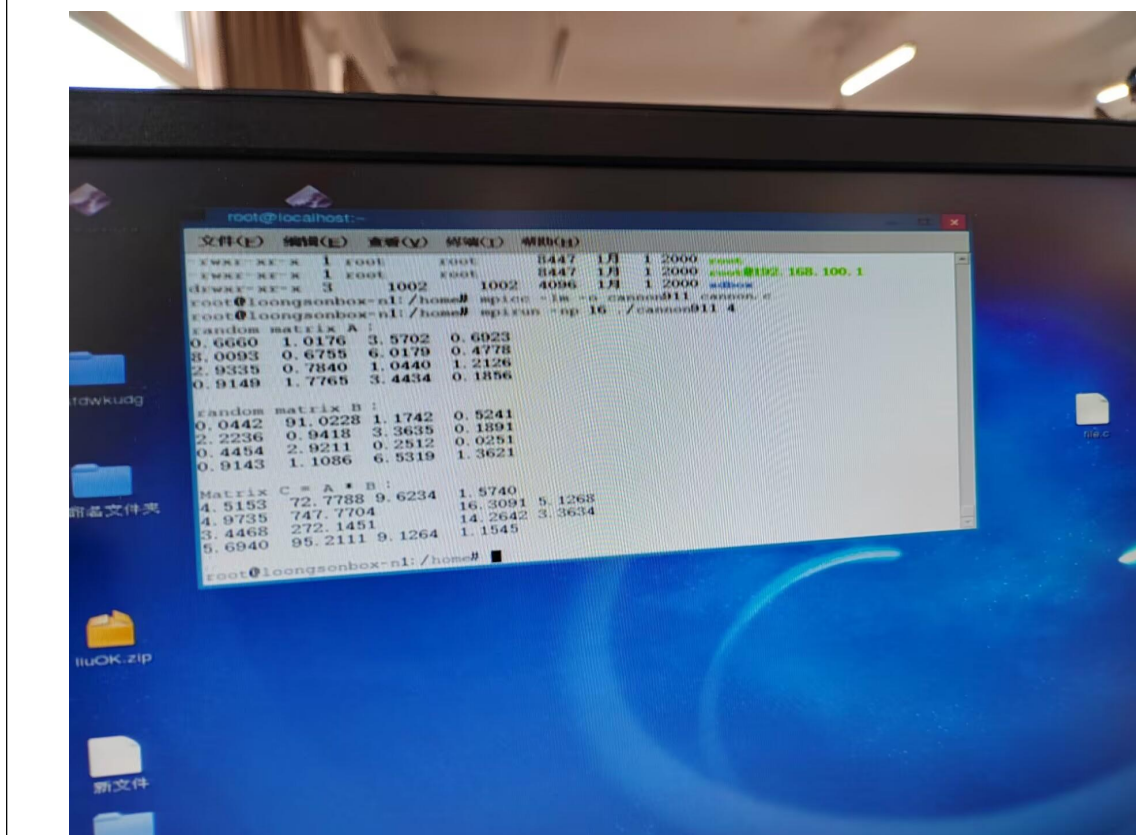
    if (my_rank == 0)
    {
        collect_C(); /* rank为0的处理单元从其他处理单元收集分块矩阵c */
        print(A, "random matrix A: \n"); /* 打印矩阵A */
        print(B, "random matrix B: \n"); /* 打印矩阵B */
        print(C, "Matrix C = A * B: \n"); /* 打印矩阵C */
    }
    else
    {
        MPI_Send(c, D12, MPI_FLOAT, 0, 1, MPI_COMM_WORLD); /* rank不为0的处理单元向rank为0的处理单元发送矩阵块c */
    }

    MPI_Barrier(MPI_COMM_WORLD); /* 同步所有处理单元 */
    MPI_Finalize(); /* 结束MPI计算 */
    return 0;
}
```

编译完成



实验结果



四、 实验过程分析与讨论

通过观察给出的代码，cannon算法是一种优化矩阵分块乘法的算法，是一种存储有效的算法，把大的矩阵划分成小的矩阵块，主要分为三个过程：分配初始位置、进行乘-加运算、循环单步移位。借助mpi通信函数实现了进程的相互通信。

结论： 随着计算节点点数/线程的增加，程序时间迅速减少，然后趋于平稳。加速比跟执行时间刚好相反变化。

五、指导教师意见

指导教师签字：卢洋