实验报告

课程：高性能计算应用实践

姓名：王峻阳

学号：220110317

学院：计算机科学与技术学院

学期：2023 年秋季学期

实验日期：2023 年 10 月 8 日

一、MPICH 多进程矩阵乘运算

一、方案介绍：

先由主进程（Rank = 0）创建矩阵, 并用随机数填充矩阵 A、B、C，然后进行多进程并行计算，最后由主进程打印计算结果和花费的时间。多进程并行计算部分，首先由主进程把三个矩阵分块分发给子进程（Rank = 1~15），接着主进程和子进程并行计算各自的矩阵块；计算结束后，子进程把各自算得的一部分矩阵 C 发送给主进程，再由主进程把各个进程的计算结果接收到矩阵 C 的正确内存位置上。各个进程计算矩阵乘法时，只使用朴素的矩阵乘算法。具体方案如下。

给定矩阵 A, B, C，计算 $C' = C + AB$。将 B 和 C 比较均匀地分块为

$$B = \begin{pmatrix} B_0 & B_1 & \cdots & B_{R-1} \end{pmatrix}, C = \begin{pmatrix} C_0 & C_1 & \cdots & C_{R-1} \end{pmatrix}$$

其中 R 是进程数，则

$$\begin{aligned} C' = C + AB &= \begin{pmatrix} C_0 & C_1 & \cdots & C_{R-1} \end{pmatrix} + A \begin{pmatrix} B_0 & B_1 & \cdots & B_{R-1} \end{pmatrix} \\ &= \begin{pmatrix} C_0 + AB_0 & C_1 + AB_1 & \cdots & C_{R-1} + AB_{R-1} \end{pmatrix} \end{aligned}$$

这样，要得到 C'，可以安排进程 i 计算 $C_i + AB_i$ $(i = 0, 1, \cdots, R-1)$，然后将每个这样的矩阵合并起来，就得到 C'。

实现方法是，主进程将 A 广播给子进程，将 B_i 和 C_i 点对点发送给从进程 i $(1 \leqslant i \leqslant 15)$，然后所有进程计算各自的 $C_i + AB_i$，计算结束后子进程将计算结果点对点发送到主进程矩阵 C 的对应内存位置上。

二、关键代码

子进程设置矩阵规模参数

```
/* Set correct arguments for child processes */
MPI_Bcast(&m, 1, MPI_INT, MAIN_PROCESS, MPI_COMM_WORLD);
MPI_Bcast(&n, 1, MPI_INT, MAIN_PROCESS, MPI_COMM_WORLD);
MPI_Bcast(&k, 1, MPI_INT, MAIN_PROCESS, MPI_COMM_WORLD);
if (my_id != MAIN_PROCESS) {
    lda = m; ldb = k; ldc = m;
}
```

子进程申请内存

```
/* Number of columns this process calculates */
int my_width = n * (my_id + 1) / num_processes - n * my_id / num_processes;
/* Leftmost columns of blocks of each process */
int n_starts[num_processes + 1];
if (my_id == MAIN_PROCESS) {
    for (int rank = 0; rank < num_processes; rank++) {
        int n_start = n * rank / num_processes;
        n_starts[rank] = n_start;
    }
    n_starts[num_processes] = n;
}
/* Allocate memory for matrices in child processes */
if (my_id != MAIN_PROCESS) {
    a = calloc(m * k, sizeof(double));
    b = calloc(k * my_width, sizeof(double));
    c = calloc(m * my_width, sizeof(double));
    if (!(a && b && c)) {
        fprintf(stderr, "Calloc failed\n");
        exit(0);
    }
}
```

主进程把任务分配给子进程

```
    /* Broadcast matrix A to child processes */
    for (int col = 0; col < k; col++) {
        MPI_Bcast(&A(0, col), m, MPI_DOUBLE, MAIN_PROCESS, MPI_COMM_WORLD);
    }
    /* Send blocks of matrices B and C to child processes */
    if (my_id == MAIN_PROCESS) {
        for (int rank = 1; rank < num_processes; rank++) {
            int block_width = n_starts[rank + 1] - n_starts[rank];
            for (int col = 0; col < block_width; col++) {
                MPI_Send(&B(0, n_starts[rank] + col), k, MPI_DOUBLE,
                         rank, 10, MPI_COMM_WORLD);
                MPI_Send(&C(0, n_starts[rank] + col), m, MPI_DOUBLE,
                         rank, 11, MPI_COMM_WORLD);
            }
        }
    }
    else {
        for (int col = 0; col < my_width; col++) {
            MPI_Status status;
            MPI_Recv(&B(0, col), k, MPI_DOUBLE,
                MAIN_PROCESS, 10, MPI_COMM_WORLD, &status);
            MPI_Recv(&C(0, col), m, MPI_DOUBLE,
                MAIN_PROCESS, 11, MPI_COMM_WORLD, &status);
        }
    }
```

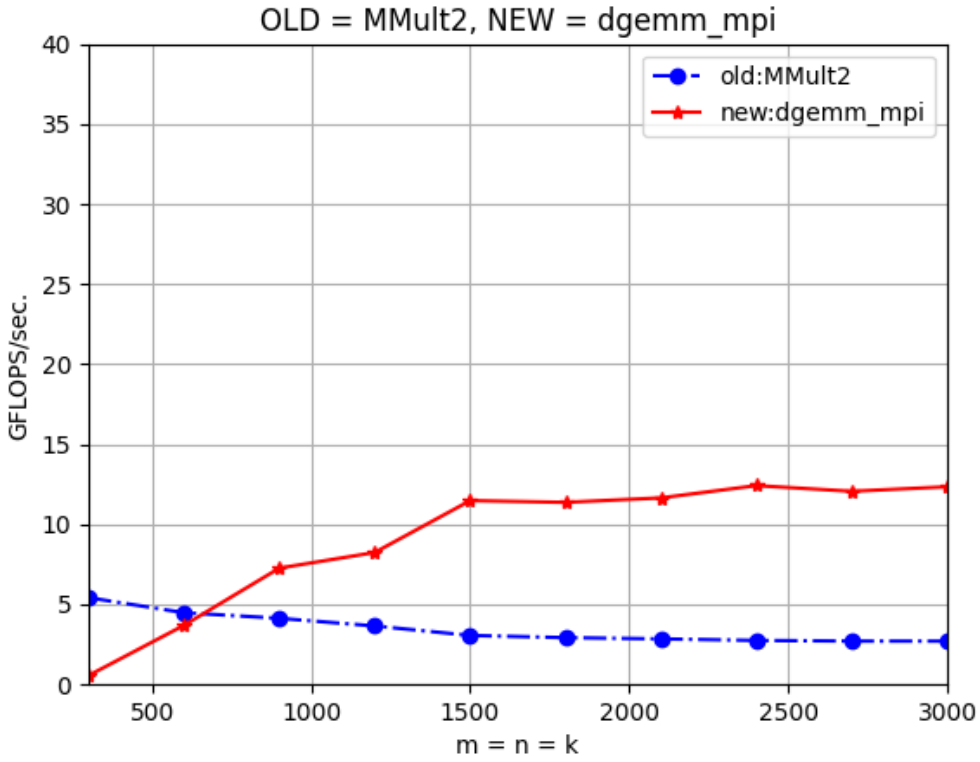
各个进程计算，然后主进程合并计算结果

```
    /* Calculate each process's own block */
    naive_dgemm(m, my_width, k, a, lda, b, ldb, c, ldc);

    /* Main process collects calculation result to matrix C */
    if (my_id == MAIN_PROCESS) {
        for (int rank = 1; rank < num_processes; rank++) {
            int block_width = n_starts[rank + 1] - n_starts[rank];
            MPI_Status status;
            for (int col = 0; col < block_width; col++) {
                MPI_Recv(&C(0, n_starts[rank] + col), m, MPI_DOUBLE,
                    rank, 12, MPI_COMM_WORLD, &status);
            }
        }
    }
    else {
        for (int col = 0; col < my_width; col++) {
            MPI_Send(&C(0, col), m, MPI_DOUBLE, MAIN_PROCESS, 12, MPI_COMM_WORLD);
        }
        free(a); free(b); free(c);
    }
}
```

三、数据分析

测算 Gflops，将本次实验的多进程朴素矩阵乘（dgemm_mpi）同朴素矩阵乘（MMult2）比较，绘制如下图表。矩阵规模设置为 m=n=k=lda=ldb=ldc；gcc 打开 O1 优化；进程数设置为 16。多进程的版本，速度能达到朴素版本的三到四倍左右。



二、遇到的问题和解决办法

问题一：与 POSIX Threads 不同，MPI 程序一开始就同时有多个线程在运行，如何控制主进程和子进程分别执行哪些指令？

参考示例程序和网上其他例子，发现只需用条件语句控制即可，尽管程序看起来比较复杂。

问题二：主进程和子进程应如何为矩阵分配内存？

从矩阵乘法的应用需求出发来想，子进程所需的空间由矩阵规模决定，为了节省内存资源，子进程可以等主进程把矩阵规模告知自己之后再申请内存。另一方面，为了省下申请内存的时间，加快运算效率，也可以先让所有进程申请足够大的内存。最后选择了第一种办法。

问题三：如何将矩阵乘运算分解，以便让每个进程独立计算，便于合并计算结果，也便于矩阵数据的发送和接收？

考虑了简单的直接按列分块，按列分块就可以满足上述需求。也考虑过先把矩阵分成小块，再把计算每小块的任务分配给子进程，但恐怕这样会产生更多的数据传输代价。

问题四：MPI 的数据传输，具体是怎样的概念？它提供的 `MPI_Bcast`、`MPI_Send` 等方法，具体做些什么？

MPICH 提供的手册比较简略，不过官网提供了简明易懂的入门教程，网上也有许多解说，很容易上手。

问题五：程序经常写着写着就 Segfault，怎么解决？

写这个程序时出问题的地方一般都是哪里数组访问越界了，哪里有笔误，或者程序逻辑有问题等等。主要还是多进程不方便调试，以后可能要学习一些多进程调试技术。