**并行计算第三次作业**

## 源码中的问题

* 矩阵的尺寸在处被写死，不利于提高计算效率。
* 号节点中，第一次循环的循环终止条件有误，将在接下来的循环中计算，因此不可取等，否则将导致计算结果出错。
* 号节点中，第一次循环时没有特判节点数多矩阵行数少，即的情况，这将导致死锁。
* 号节点中，第二次循环用于计算的行数错误，应为这将导致计算结果出错。
* 上述问题在代码注释中有更详细的解释。

## 程序流程图

## 程序解决方案

* 每一个计算节点启动一个进程。
* 进程内启动多个线程，其中一个线程处理通信。

## 注释后的实验源程序

**#include <mpi.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <pthread.h>**

**#include <sys/stat.h>**

**#include <sys/sysinfo.h>**

**/\***

**\* 定义矩阵的大小**

**\* 计算加速比时采用较大的居中，以便得到清晰稳定的区别**

**\*/**

**#define LEN\_N 2100**

**#define LEN\_M 2100**

**#define LEN\_P 2100**

**/\***

**\* 将计算最小值的过程封装**

**\* 由于比较简单，采用宏函数实现**

**\*/**

**#define min(a, b) (((a) < (b)) ? (a) : (b))**

**/\***

**\* 用于计算的三个矩阵**

**\* 由于矩阵较大，从栈空间移到全局变量区**

**\*/**

**double A[LEN\_N][LEN\_M], B[LEN\_M][LEN\_P], C[LEN\_N][LEN\_P];**

**/\***

**\* 定义线程的参数**

**\* 将传参封装成一个结构体**

**\*/**

**struct threadArg {**

**/\***

**\* tid：线程的 id**

**\* A\_row：相乘行指针**

**\* C\_row：结果行指针**

**\* num\_threads：线程个数**

**\* len\_m：A\_row 的真实长度(而不是理论长度 LEN\_M )**

**\* len\_p：C\_row 的真实长度(而不是理论长度 LEN\_P )**

**\* 由于 B 已经移动到全局区，不需要通过参数传递**

**\*/**

**int tid;**

**double\* A\_row;**

**double\* C\_row;**

**int num\_threads;**

**int len\_m, len\_p;**

**};**

**/\***

**\* 实际计算的过程**

**\* 完成结果行的计算**

**\*/**

**void\* worker(void\* arg) {**

**struct threadArg\* my\_arg = (struct threadArg\*)arg;**

**// printf("m = %d, p = %d\n", my\_arg->len\_m, my\_arg->len\_p);**

**/\***

**\* 此处不需要开多少枚举多少，只需要用多少枚举多少，因此**

**\* 将理论长度 LEN\_P 改为真实长度 len\_p 枚举**

**\* 将理论长度 LEN\_M 改为真实长度 len\_m 枚举**

**\* 这种方式下(对于小矩阵)性能将大大提升**

**\***

**\* 源码中的问题1：**

**\* 矩阵的尺寸在 LEN\_N LEN\_M LEN\_P 处被写死，不利于提高计算效率**

**\*/**

**for (int i = my\_arg->tid; i < my\_arg->len\_p; i += my\_arg->num\_threads) {**

**/\***

**\* 动态申请的空间在堆中**

**\* 由于初值是不确定的，使用前需要先清空**

**\*/**

**my\_arg->C\_row[i] = 0.0;**

**for (int j = 0; j < my\_arg->len\_m; j++) {**

**/\***

**\* 串行的枚举完成计算即可**

**\* 将 B 中第 j 行第 i 个元素乘 A 对应的第 j 个元素**

**\* 累加到 C 对应的第 i 个元素**

**\*/**

**my\_arg->C\_row[i] += my\_arg->A\_row[j] \* B[j][i];**

**}**

**}**

**return NULL;**

**}**

**/\***

**\* 读文件过程，修改了 matrix 的内存使用**

**\* 具体的， matrix 每一行的存储长度是 width 以便使用 matrix[i] 访问**

**\* matrix 实际的大小则通过 n 和 m 作为结果输出**

**\*/**

**int read\_file(const char\* filename, double\* matrix, const int width, int\* n, int\* m) {**

**/\***

**\* 开文件并特判**

**\* 文件不存在直接报错并退出**

**\*/**

**FILE\* file\_ptr;**

**if (!(file\_ptr = fopen(filename, "r"))) {**

**printf("Can't open file %s\n", filename);**

**exit(-1);**

**}**

**/\***

**\* 获取文件大小并将文件一次性读入内存**

**\* 从文件前两个字获取矩阵的尺寸信息**

**\*/**

**struct stat f\_stat;**

**stat(filename, &f\_stat);**

**int f\_size = f\_stat.st\_size;**

**char\* f\_stream = (char\*)malloc(f\_size);**

**fread(f\_stream, sizeof(char), f\_size, file\_ptr);**

**\*n = ((int\*)f\_stream)[0], \*m = ((int\*)f\_stream)[1];**

**/\***

**\* 处理异常情况：**

**\* 矩阵的长宽必须是正数**

**\*/**

**if (\*n <= 0 || \*m <= 0) {**

**printf("Matrix size error, %dx%d\n", \*n, \*m);**

**exit(-1);**

**}**

**/\***

**\* 处理异常情况**

**\* 矩阵与其尺寸信息的总和不应超过文件大小**

**\*/**

**if (f\_size < (sizeof(int) \* 2 + sizeof(double) \* \*n \* \*m)) {**

**printf("Actual size mismatches with stated size\n");**

**exit(-1);**

**}**

**/\***

**\* 此处原本是 matrix[i \* \*m + j] = matrix\_addr[i \* \*m + j] 这样可以最小化矩阵占用的空间**

**\* 但矩阵已经以数组的形式开出，实际空间总在编译阶段确定，因此可以改用 matrix[i \* width + j]**

**\* 使用 matrix[i \* width + j] 的好处在于与 C 语言实际存储二维数组的方式一致**

**\* 因此可以方便的使用 matrix[i] 访问其第 i 行**

**\*/**

**double\* matrix\_addr = (double\*)(f\_stream + sizeof(int) \* 2);**

**printf(" ---- %s: %d \* %d Matrix -----\n", filename, \*n, \*m);**

**for (int i = 0; i < \*n; i++) {**

**for (int j = 0; j < \*m; j++) {**

**matrix[i \* width + j] = matrix\_addr[i \* \*m + j];**

**}**

**}**

**free(f\_stream), fclose(file\_ptr);**

**return 0;**

**}**

**/\***

**\* 写文件过程，由于修改了 matrix 的内存使用，需要同时修改输出方式**

**\* 具体的，存储到文件的矩阵每一行不存在空间冗余**

**\*/**

**int write\_file(const char\* filename, const int n, const int m, const int width, const double\* matrix) {**

**/\***

**\* 分配存储空间，并记录矩阵信息**

**\* 存储到文件的矩阵，其最开始两个字分别表示其行数和列数**

**\* 接下来行数乘列数个字按行存储整个矩阵**

**\*/**

**int buf\_size = (int)(sizeof(int) \* 2 + sizeof(double) \* n \* m);**

**double\* buffer = (double\*)malloc(buf\_size);**

**((int\*)buffer)[0] = n;**

**((int\*)buffer)[1] = m;**

**double\* ptr = (double\*)((int\*)buffer + 2);**

**printf("Result matrix: %d \* %d\n", n, m);**

**for (int i = 0; i < n; i++) {**

**for (int j = 0; j < m; j++) {**

**/\***

**\* 输出矩阵信息并写入缓存**

**\* 读取时的处理为 matrix[i \* \*m + j] = matrix\_addr[i \* \*m + j]**

**\* 写入时的处理为 ptr[i \* m + j] = matrix[i \* width + j]**

**\* 二者恰好相反**

**\*/**

**printf("%.4f ", matrix[i \* width + j]);**

**ptr[i \* m + j] = matrix[i \* width + j];**

**}**

**putchar('\n');**

**}**

**FILE\* file\_ptr;**

**/\***

**\* 开文件并特判**

**\* 文件不存在应直接报错并退出，而不是只报错不退出**

**\* 否则会因空指针出现 run time error**

**\*/**

**if (!(file\_ptr = fopen(filename, "w"))) {**

**printf("Can't open file %s\n", filename);**

**exit(-1);**

**}**

**fwrite(buffer, sizeof(char), buf\_size, file\_ptr);**

**fclose(file\_ptr);**

**return 0;**

**}**

**/\***

**\* 将运行时间(不考虑 IO 时间)输出到文件**

**\* 由于写结果的文件和标准输出均已经存在内容，另开一个文件写时间**

**\*/**

**int write\_time(const char\* filename, const double time) {**

**FILE\* file\_ptr;**

**/\***

**\* 开文件并特判**

**\* 文件不存在应直接报错并退出，而不是只报错不退出**

**\* 否则会因空指针出现 run time error**

**\*/**

**if (!(file\_ptr = fopen(filename, "w"))) {**

**printf("Can't open file %s\n", filename);**

**exit(-1);**

**}**

**fprintf(file\_ptr, "Total time is %lf\n", time);**

**fclose(file\_ptr);**

**return 0;**

**}**

**int main(int argc, char\* argv[]) {**

**/\***

**\* 记录矩阵实际的尺寸**

**\* 三个矩阵的实际尺寸依次是 n x m, m x p, n x p**

**\*/**

**int n, m, p;**

**/\***

**\* 初始化 MPI 环境**

**\* status 用于获取 MPI\_TAG**

**\*/**

**MPI\_Status status;**

**int my\_id, num\_proc;**

**MPI\_Init(&argc, &argv);**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_id);**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_proc);**

**if (!my\_id) {**

**/\***

**\* 0 号节点读矩阵文件**

**\* 读矩阵的同时完成 0 号节点中，对矩阵尺寸的更新**

**\*/**

**read\_file("matrix\_a.stdin", (double\*)A, LEN\_M, &n, &m);**

**read\_file("matrix\_b.stdin", (double\*)B, LEN\_P, &m, &p);**

**}**

**/\***

**\* MPI\_Bcast广播发送消息，将 root 的 buffer 广播到 comm 域**

**\* 由于每个节点都有收和发的操作，需要将bcast代码放在所有程序都能运行到的地方**

**\*/**

**MPI\_Bcast(B, LEN\_M \* LEN\_P, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**/\***

**\* 除了矩阵 B 之外，广播矩阵的尺寸**

**\* 函数 void\* worker(void\* arg) 中将使用 m 和 p**

**\*/**

**MPI\_Bcast(&m, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Bcast(&p, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**if (!my\_id) {**

**/\***

**\* IO 完成，在主节点(即 0 号节点)开始计时**

**\* 确定第一轮发送的数目，即取 MPI 节点数目和矩阵行数的较小者**

**\*/**

**double start\_time = MPI\_Wtime();**

**int num\_send = min(num\_proc - 1, n);**

**for (int i = 1; i < num\_proc; i++) {**

**/\***

**\* 源码中的问题2：**

**\* 循环终止条件有误，A[num\_proc] 将在接下来的循环中计算**

**\* 因此 i < num\_proc 不取等，否则将导致计算结果出错**

**\***

**\* 向第 i 个节点发送矩阵的 i - 1 行**

**\* 需要注意，当节点多矩阵行数少的时候，需要向多余的节点发送结束消息**

**\* 否则这些节点将一直卡在 MPI\_Recv 上，即死锁**

**\***

**\* 源码中的问题3：**

**\* 没有特判节点数多矩阵行数少的情况，将导致死锁**

**\*/**

**if (i <= num\_send) {**

**MPI\_Send(A[i - 1], m, MPI\_DOUBLE, i, 99, MPI\_COMM\_WORLD);**

**}**

**else {**

**int dev\_null;**

**MPI\_Send(&dev\_null, 0, MPI\_INT, i, 0, MPI\_COMM\_WORLD);**

**}**

**}**

**for (int i = 1; i <= n; i++) {**

**int sender = (i - 1) % (num\_proc - 1) + 1;**

**/\***

**\* 循环接收每一行，如果某一行计算完之后还有行**

**\* 就将这一行分配，继续计算，否则发送结束信号**

**\* 需要注意的是，原程序中 MPI\_Send(A[num\_send - 1], m, MPI\_DOUBLE, sender, 99, MPI\_COMM\_WORLD); 有误**

**\* 在上一阶段的循环中 num\_send - 1 及之前的部分全部计算完成，因此要从 num\_send 开始计算**

**\***

**\* 源码中的问题4：**

**\* 用于计算的行数错误，将导致计算结果出错**

**\*/**

**MPI\_Recv(C[i - 1], p, MPI\_DOUBLE, sender, 100, MPI\_COMM\_WORLD, &status);**

**if (num\_send < n) {**

**MPI\_Send(A[num\_send], m, MPI\_DOUBLE, sender, 99, MPI\_COMM\_WORLD);**

**num\_send++;**

**}**

**else {**

**int dev\_null;**

**MPI\_Send(&dev\_null, 0, MPI\_INT, sender, 0, MPI\_COMM\_WORLD);**

**}**

**}**

**/\***

**\* 计算完成，停止计时(和串行程序一样，不计算 IO 时间)**

**\* 分别输出结果和时间信息到两个文件**

**\*/**

**double finish\_time = MPI\_Wtime();**

**write\_time("run\_time.stdout", finish\_time - start\_time);**

**write\_file("result.stdout", n, p, LEN\_P, C[0]);**

**}**

**else {**

**/\***

**\* 根据处理器核心数设置线程数**

**\* 为每个线程分配存储线程号的空间**

**\* 为用于计算的行和存储结果的行分配空间**

**\* 为传给各个线程的参数分配空间**

**\*/**

**int num\_threads = get\_nprocs();**

**pthread\_t\* thread\_ids = (pthread\_t\*)malloc(num\_threads \* sizeof(pthread\_t));**

**double\* A\_row = (double\*)malloc(m \* sizeof(double));**

**double\* C\_row = (double\*)malloc(p \* sizeof(double));**

**struct threadArg\* thread\_args = (struct threadArg \*)malloc(num\_threads \* sizeof(struct threadArg));**

**for (int i = 0; i < num\_threads; i++) {**

**/\***

**\* 初始化线程 ID 用于划分任务**

**\* 初始化总线程个数用于划分任务**

**\* 传入的参数 m 和 p 在此处设置**

**\*/**

**thread\_args[i].tid = i;**

**thread\_args[i].len\_m = m;**

**thread\_args[i].len\_p = p;**

**thread\_args[i].A\_row = A\_row;**

**thread\_args[i].C\_row = C\_row;**

**thread\_args[i].num\_threads = num\_threads;**

**}**

**while (1) {**

**/\***

**\* 循环接收消息，若收到结束信号(即 status.MPI\_TAG == 0 时)则停止**

**\* 否则创建若干线程计算出 C\_row 并返回**

**\*/**

**MPI\_Recv(A\_row, m, MPI\_DOUBLE, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);**

**if (status.MPI\_TAG == 0) {**

**break;**

**}**

**/\***

**\* 先统一创建各个线程并开始执行**

**\* 再等待各个线程全部结束**

**\* 处理完成，将结果发送回 0 号节点**

**\*/**

**for (int i = 0; i < num\_threads; i++) {**

**pthread\_create(thread\_ids + i, NULL, worker, thread\_args + i);**

**}**

**for (int i = 0; i < num\_threads; i++) {**

**pthread\_join(thread\_ids[i], NULL);**

**}**

**MPI\_Send(C\_row, p, MPI\_DOUBLE, 0, 100, MPI\_COMM\_WORLD);**

**}**

**}**

**/\***

**\* 结束 MPI 环境并退出程序**

**\*/**

**MPI\_Finalize();**

**return 0;**

**}**

## 辅助程序：生成矩阵(进行了格式化及调整)

**#include <time.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main(int argc, char\* argv[]) {**

**if (argc < 4) {**

**printf("Invalid arguments!\n");**

**printf("Run the program as ./gen\_matrix n m filename\n");**

**printf("n, m are row/col number of the matrix, filename is the file to write\n");**

**return 0;**

**}**

**char\* filename = argv[3];**

**int n = (int)strtol(argv[1], NULL, 10);**

**int m = (int)strtol(argv[2], NULL, 10);**

**int buf\_size = (int)(sizeof(int) \* 2 + sizeof(double) \* n \* m);**

**double\* matrix = (double\*)malloc(buf\_size);**

**((int\*)matrix)[0] = n;**

**((int\*)matrix)[1] = m;**

**double\* ptr = (double\*)((int\*)matrix + 2);**

**srand(n \* m \* 0x19260817);**

**for (int i = 0; i < n; i++) {**

**for (int j = 0; j < m; j++) {**

**ptr[i \* m + j] = (double)random() / RAND\_MAX;**

**}**

**}**

**FILE\* file\_ptr;**

**if (!(file\_ptr = fopen(filename, "w"))) {**

**printf("Can't open file %s\n", filename);**

**exit(-1);**

**}**

**fwrite(matrix, sizeof(char), buf\_size, file\_ptr);**

**fclose(file\_ptr);**

**return 0;**

**}**

## 辅助程序：输出矩阵(进行了格式化及调整)

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <sys/stat.h>**

**int main(int argc, char\* argv[]) {**

**FILE\* file\_ptr;**

**if (argc < 2) {**

**printf("Invalid arguments!\n");**

**printf("Run the program as ./print filename\n");**

**exit(-1);**

**}**

**if (!(file\_ptr = fopen(argv[1], "r"))) {**

**printf("Can't open file %s\n", argv[1]);**

**exit(-1);**

**}**

**struct stat f\_stat;**

**stat(argv[1], &f\_stat);**

**int f\_size = f\_stat.st\_size;**

**char\* f\_stream = (char \*)malloc(f\_size);**

**fread(f\_stream, sizeof(char), f\_size, file\_ptr);**

**int n = ((int\*)f\_stream)[0], m = ((int\*)f\_stream)[1];**

**double\* matrix = (double\*)(f\_stream + sizeof(int) \* 2);**

**if (n <= 0 || m <= 0) {**

**printf("Matrix size error, %dx%d\n", n, m);**

**exit(-1);**

**}**

**if (f\_size < (sizeof(int) \* 2 + sizeof(double) \* n \* m)) {**

**printf("Actual size mismatches with stated size\n");**

**exit(-1);**

**}**

**printf(" ---- %s: %d \* %d Matrix -----\n", argv[1], n, m);**

**for (int i = 0; i < n; i++) {**

**for (int j = 0; j < m; j++) {**

**printf("%.4f ", matrix[i \* m + j]);**

**}**

**putchar('\n');**

**}**

**free(f\_stream), fclose(file\_ptr);**

**return 0;**

**}**

## 辅助程序：对比矩阵(进行了格式化及调整)

**#include <math.h>**

**#include <errno.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <sys/stat.h>**

**// Preprocess the command line, read in matrix A, B from input files, allocate**

**// memory for buffers, i.e, f\_stream\_a, f\_stream\_b to cache them.**

**int setup(const int argc, char\* argv[], char\*\* f\_stream\_a, char\*\* f\_stream\_b) {**

**if (argc < 3) {**

**printf("Invalid arguments!\n");**

**printf("Usage: ./serial filea fileb\n");**

**printf("filea, fileb are file names for matrix A, B to be compared\n");**

**return 1;**

**}**

**FILE\* file\_ptr\_a;**

**FILE\* file\_ptr\_b;**

**if (!(file\_ptr\_a = fopen(argv[1], "r"))) {**

**printf("Can't open matrix file %s, Errno = %d\n", argv[1], errno);**

**return 1;**

**}**

**if (!(file\_ptr\_b = fopen(argv[2], "r"))) {**

**printf("Can't open matrix file %s, Errno = %d\n", argv[2], errno);**

**return 1;**

**}**

**struct stat f\_stat\_a, f\_stat\_b;**

**stat(argv[1], &f\_stat\_a);**

**stat(argv[2], &f\_stat\_b);**

**int f\_size\_a = f\_stat\_a.st\_size;**

**int f\_size\_b = f\_stat\_b.st\_size;**

**\*f\_stream\_a = (char\*)malloc(f\_size\_a);**

**\*f\_stream\_b = (char\*)malloc(f\_size\_b);**

**fread(\*f\_stream\_a, sizeof(char), f\_size\_a, file\_ptr\_a);**

**fread(\*f\_stream\_b, sizeof(char), f\_size\_b, file\_ptr\_b);**

**int n1 = ((int\*)\*f\_stream\_a)[0], m1 = ((int\*)\*f\_stream\_a)[1];**

**int n2 = ((int\*)\*f\_stream\_b)[0], m2 = ((int\*)\*f\_stream\_b)[1];**

**if (n1 <= 0 || m1 <= 0 || n2 <= 0 || m2 <= 0) {**

**printf("Matrix size error, %dx%d with %dx%d\n", n1, m1, n2, m2);**

**return 1;**

**}**

**if (f\_size\_a < sizeof(int) \* 2 + sizeof(double) \* n1 \* m1) {**

**printf("Actual size of A mismatches with stated size\n");**

**return 1;**

**}**

**if (f\_size\_b < sizeof(int) \* 2 + sizeof(double) \* n2 \* m2) {**

**printf("Actual size of B mismatches with stated size\n");**

**return 1;**

**}**

**fclose(file\_ptr\_a);**

**fclose(file\_ptr\_b);**

**return 0;**

**}**

**void comp(const char\* f\_stream\_a, const char\* f\_stream\_b) {**

**int n1 = ((int\*)f\_stream\_a)[0], m1 = ((int\*)f\_stream\_a)[1];**

**int n2 = ((int\*)f\_stream\_b)[0], m2 = ((int\*)f\_stream\_b)[1];**

**if (n1 != n2 || m1 != m2) {**

**printf("Matrix size mismatch, %dx%d with %dx%d\n", n1, m1, n2, m2);**

**return;**

**}**

**double norm = 0.0;**

**double\* A = (double\*)(f\_stream\_a + sizeof(int) \* 2);**

**double\* B = (double\*)(f\_stream\_b + sizeof(int) \* 2);**

**for (int i = 0; i < n1; i++) {**

**for (int j = 0; j < m1; j++) {**

**double diff = A[i \* m1 + j] - B[i \* m1 + j];**

**norm += diff \* diff;**

**}**

**}**

**norm = sqrt(norm);**

**if (norm > 0.000001) {**

**printf("Matrix compare failed, with norm = %.8f\n", norm);**

**}**

**else {**

**printf("Matrix compare succeeded, with norm = %.8f\n", norm);**

**}**

**}**

**int main(int argc, char\* argv[]) {**

**// Buffers to cache matrix files of A, B**

**char\* f\_stream\_a;**

**char\* f\_stream\_b;**

**// preprocess the command line, read in files for A, B.**

**if (setup(argc, argv, &f\_stream\_a, &f\_stream\_b)) {**

**// Something error during pre processing**

**exit(-1);**

**}**

**comp(f\_stream\_a, f\_stream\_b);**

**free(f\_stream\_a);**

**free(f\_stream\_b);**

**return 0;**

**}**

## 辅助程序：串行矩阵乘法(进行了格式化及调整)

**#include <errno.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/stat.h>**

**#include <sys/time.h>**

**double w\_time() {**

**struct timeval tv;**

**gettimeofday(&tv, NULL);**

**return tv.tv\_sec + 1E-6 \* tv.tv\_usec;**

**}**

**// Preprocess the command line, read in matrix A, B from input files, allocate**

**// memory for buffers, i.e, f\_stream\_a, f\_stream\_b to cache them. Suppose A, B's**

**// size are n1 \* n2, n2 \* n3, then n1 ~ n3 will be stored at dim[0 ~ 2]**

**// Return value 0 means no error occurred during pre processing, otherwise a**

**// non-zero returns.**

**int setup(int argc, char\* argv[], char\*\* f\_stream\_a, char\*\* f\_stream\_b, int\* dim) {**

**if (argc < 4) {**

**printf("Invalid arguments!\n");**

**printf("Usage: ./serial filea fileb filec\n");**

**printf("filea, fileb and filec are file names for matrix A, B and C\n");**

**return 1;**

**}**

**FILE\* file\_ptr\_a;**

**FILE\* file\_ptr\_b;**

**if (!(file\_ptr\_a = fopen(argv[1], "r"))) {**

**printf("Can't open matrix file %s, Errno=%d\n", argv[1], errno);**

**return 1;**

**}**

**if (!(file\_ptr\_b = fopen(argv[2], "r"))) {**

**printf("Can't open matrix file %s, Errno=%d\n", argv[2], errno);**

**return 1;**

**}**

**struct stat f\_stat\_a, f\_stat\_b;**

**stat(argv[1], &f\_stat\_a);**

**stat(argv[2], &f\_stat\_b);**

**int f\_size\_a = f\_stat\_a.st\_size;**

**int f\_size\_b = f\_stat\_b.st\_size;**

**\*f\_stream\_a = (char\*)malloc(f\_size\_a);**

**\*f\_stream\_b = (char\*)malloc(f\_size\_b);**

**fread(\*f\_stream\_a, sizeof(char), f\_size\_a, file\_ptr\_a);**

**fread(\*f\_stream\_b, sizeof(char), f\_size\_b, file\_ptr\_b);**

**int n1 = ((int\*)\*f\_stream\_a)[0], m1 = ((int\*)\*f\_stream\_a)[1];**

**int n2 = ((int\*)\*f\_stream\_b)[0], m2 = ((int\*)\*f\_stream\_b)[1];**

**if (n1 <=0 || m1 <= 0 || n2 <= 0 || m2 <= 0 || m1 != n2) {**

**printf("Matrix size error, %dx%d with %dx%d\n", n1, m1, n2, m2);**

**return 1;**

**}**

**if (f\_size\_a < sizeof(int) \* 2 + sizeof(double) \* n1 \* m1) {**

**printf("Actual size of A mismatches with stated size\n");**

**return 1;**

**}**

**if (f\_size\_b < sizeof(int) \* 2 + sizeof(double) \* n2 \* m2) {**

**printf("Actual size of B mismatches with stated size\n");**

**return 1;**

**}**

**dim[0] = n1, dim[1] = m1, dim[2] = m2;**

**fclose(file\_ptr\_a), fclose(file\_ptr\_b);**

**return 0;**

**}**

**// Compute C = A \* B. A is a n1 \* n2 matrix. B is a n2 \* n3 matrix.**

**void matmul(const double\* A, const double\* B, double\* C, const int n1, const int n2, const int n3) {**

**for (int i = 0; i < n1; i++) {**

**for (int j = 0; j < n3; j++) {**

**C[i \* n3 + j] = 0.0;**

**for (int k = 0; k < n2; k++) {**

**C[i \* n3 + j] += A[i \* n2 + k] \* B[k \* n3 + j];**

**}**

**}**

**}**

**}**

**int main(int argc, char\* argv[]) {**

**// Buffers to cache matrix files of A, B**

**char\* f\_stream\_a;**

**char\* f\_stream\_b;**

**// Preprocess the command line, read in files for**

**// A, B and put their sizes in dim[].**

**int dim[3];**

**if (setup(argc, argv, &f\_stream\_a, &f\_stream\_b, dim)) {**

**// Something error during pre processing**

**exit(-1);**

**}**

**// Suppose A's size is n1 x n2, B's is n2 x n3.**

**// n1 ~ n3 are read from input files.**

**int n1 = dim[0];**

**int n2 = dim[1];**

**int n3 = dim[2];**

**FILE\* file\_ptr;**

**int f\_size\_c = (int)(sizeof(int) \* 2 + sizeof(double) \* n1 \* n3);**

**if (!(file\_ptr = fopen(argv[3], "w"))) {**

**printf("Can't open file %s, Errno=%d\n", argv[3], errno);**

**exit(-1);**

**}**

**char\* f\_stream\_c = (char \*)malloc(f\_size\_c);**

**((int\*)f\_stream\_c)[0] = n1;**

**((int\*)f\_stream\_c)[1] = n3;**

**double elapsed\_time = w\_time();**

**matmul((double\*)(f\_stream\_a + sizeof(int) \* 2),**

**(double\*)(f\_stream\_b + sizeof(int) \* 2),**

**(double\*)(f\_stream\_c + sizeof(int) \* 2),**

**n1, n2, n3);**

**elapsed\_time = w\_time() - elapsed\_time;**

**printf("Serial algorithm: multiply a %d x %d with a %d x %d, use %.2f(s)\n",**

**n1, n2, n2, n3, elapsed\_time);**

**fwrite(f\_stream\_c, sizeof(char), f\_size\_c, file\_ptr);**

**free(f\_stream\_a);**

**free(f\_stream\_b);**

**free(f\_stream\_c);**

**fclose(file\_ptr);**

**return 0;**

**}**

## 自动评测工具：生成+两种方式计算+输出结果

**import os**

**import time**

**OPTIONS = ['-lm', '-std=c99', '-Wall', '-O2']**

**COMPLIE\_LIST = [**

**'compare',**

**'gen\_matrix',**

**'mult\_serial',**

**'print'**

**]**

**if \_\_name\_\_ == '\_\_main\_\_':**

**if os.path.exists('result.stdout'):**

**os.remove('result.stdout')**

**os.system('mpicc mult\_matrix.c -o mult\_matrix ' + ' '.join(OPTIONS))**

**for i in COMPLIE\_LIST:**

**print('compiling ' + i)**

**os.system('gcc ' + i + '.c -o ' + i + ' ' + ' '.join(OPTIONS))**

**n, m, p = input().split()**

**os.system('./gen\_matrix ' + n + ' ' + m + ' ' + 'matrix\_a.stdin')**

**os.system('./gen\_matrix ' + m + ' ' + p + ' ' + 'matrix\_b.stdin')**

**if int(n) \* int(m) \* int(p) < 2048:**

**os.system('./print matrix\_a.stdin')**

**print()**

**os.system('./print matrix\_b.stdin')**

**print()**

**os.system('./mult\_serial matrix\_a.stdin matrix\_b.stdin standard.stdout')**

**print()**

**if int(n) \* int(m) \* int(p) < 2048:**

**os.system('./print standard.stdout')**

**print()**

**os.system('sbatch run.sh')**

**print()**

**retry = 0**

**while retry < 20 and not os.path.exists('result.stdout'):**

**time.sleep(0.5)**

**retry += 1**

**if retry == 20:**

**print('Failed to read result.stdout')**

**exit(0)**

**if int(n) \* int(m) \* int(p) < 2048:**

**os.system('./print result.stdout')**

**print()**

**os.system('./compare standard.stdout result.stdout')**

**print()**

## 实验结果

使用小矩阵验证正确性，会根据判断矩阵大小，只输出小矩阵。

**compiling compare**

**compiling gen\_matrix**

**compiling mult\_serial**

**compiling print**

**3 5 4**

**---- matrix\_a.stdin: 3 \* 5 Matrix -----**

**0.9004 0.4363 0.3523 0.5032 0.3830**

**0.8391 0.9089 0.0087 0.2009 0.2086**

**0.3505 0.9459 0.7904 0.0259 0.2680**

**---- matrix\_b.stdin: 5 \* 4 Matrix -----**

**0.4086 0.8831 0.0867 0.9339**

**0.6212 0.5153 0.4247 0.3120**

**0.0560 0.8646 0.6538 0.9129**

**0.2245 0.8379 0.9091 0.5485**

**0.5466 0.6336 0.9714 0.3646**

**Serial algorithm: multiply a 3 x 5 with a 5 x 4, use 0.00(s)**

**---- standard.stdout: 3 \* 4 Matrix -----**

**0.9810 1.9889 1.3232 1.7143**

**1.0670 1.5174 0.8497 1.2614**

**0.9274 1.6719 1.2328 1.4559**

**Submitted batch job 1255337**

**---- result.stdout: 3 \* 4 Matrix -----**

**0.9810 1.9889 1.3232 1.7143**

**1.0670 1.5174 0.8497 1.2614**

**0.9274 1.6719 1.2328 1.4559**

**Matrix compare succeeded, with norm = 0.00000000**

使用大矩阵测试性能，此矩阵过大故不会输出。

**compiling compare**

**compiling gen\_matrix**

**compiling mult\_serial**

**compiling print**

**1200 1200 1200**

**Serial algorithm: multiply a 1200 x 1200 with a 1200 x 1200, use 3.37(s)**

**Submitted batch job 1255338**

**Matrix compare succeeded, with norm = 0.00000000**

从文件读取并行程序的运行时间，可得加速比为，其中串行程序和并行程序均开启了编译优化以减小常数使结果更为客观。

**[pc19373354@ln01 homework3]$ cat run\_time.stdout**

**Total time is** **0.392884**