**并行计算第四次作业**

## 程序流程图

## 注释后的实验源程序

**#include <mpi.h>**

**#include <math.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

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

**/\***

**\* 定义节点间通信使用的 TAG**

**\* SHIFT\_TAG 用于合并结果时的移动**

**\* SCATTER\_A\_TAG 和 SCATTER\_B\_TAG 分别在拆分任务和进行计算时使用**

**\* SCATTER\_A\_TAG 始终用于区分或处理矩阵 A**

**\* SCATTER\_B\_TAG 始终用于区分或处理矩阵 B**

**\*/**

**#define SHIFT\_TAG 77**

**#define SCATTER\_A\_TAG 88**

**#define SCATTER\_B\_TAG 99**

**/\***

**\* 将多个函数公用的变量提取为全局变量**

**\* 这种处理在减少调用传参时压栈开销的同时**

**\* 有利于编译器将变量优化为寄存器变量，从而提升性能**

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

**\*/**

**int my\_id, num\_proc, n, m, p;**

**int rooted\_proc, max\_rows\_a, max\_cols\_a, max\_rows\_b, max\_cols\_b;**

**/\***

**\* 交换两个 double\* 类型数据**

**\* 一个更一般性的实现是采用 void\* 类型的参数，使用 memcpy 完成交换**

**\* 考虑到本程序中只需要交换两个 double\* 类型的数据，采用以下更简单高效的实现**

**\*/**

**void swap(double\*\* x, double\*\* y) {**

**double\* temp = \*x;**

**\*x = \*y, \*y = temp;**

**}**

**/\***

**\* 计算 (x, y) 向上循环移动 shift 个单位后的一维偏移**

**\* 采用作差对 rooted\_proc 取模加 rooted\_proc 再取模一次的方式**

**\* 可以保证取模结果处于区间 [0, rooted\_proc) 而不会出现负数**

**\*/**

**int shift\_up(const int x, const int y, const int shift) {**

**return ((x - shift) % rooted\_proc + rooted\_proc) % rooted\_proc \* rooted\_proc + y;**

**}**

**/\***

**\* 计算 (x, y) 向左循环移动 shift 个单位后的一维偏移**

**\* 采用作差对 rooted\_proc 取模加 rooted\_proc 再取模一次的方式**

**\* 可以保证取模结果处于区间 [0, rooted\_proc) 而不会出现负数**

**\*/**

**int shift\_left(const int x, const int y, const int shift) {**

**return x \* rooted\_proc + ((y - shift) % rooted\_proc + rooted\_proc) % rooted\_proc;**

**}**

**/\***

**\* 分发矩阵，通过 tag 对矩阵进行区分**

**\* tag 为 SCATTER\_A\_TAG 时处理 A 矩阵**

**\* tag 为 SCATTER\_B\_TAG 时处理 B 矩阵**

**\* row 和 col 分别对应矩阵的行数和列数**

**\*/**

**void scatter\_matrix(double\* buf, const int row, const int col, const int tag, const double\* buffer) {**

**/\***

**\* 计算最大行数和最大列数**

**\* 缓存行数列数的乘积以提高程序性能**

**\*/**

**int max\_rows = (row + rooted\_proc - 1) / rooted\_proc;**

**int max\_cols = (col + rooted\_proc - 1) / rooted\_proc;**

**int total = max\_rows \* max\_cols;**

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

**/\***

**\* 只申请 buffer 一次，反复使用后释放**

**\* 避免多次动态申请内存造成程序性能下降**

**\*/**

**double\* temp = (double\*)malloc(sizeof(double) \* total);**

**for (int i = 0; i < rooted\_proc; i++) for (int j = 0; j < rooted\_proc; j++) {**

**/\***

**\* 遍历 i 和 j 并处理 (i, j) 节点对应的数据块**

**\* 对于不能整除时出现的空间冗余，将多余的位置赋 0 处理**

**\*/**

**for (int k = 0; k < max\_rows; k++) for (int l = 0; l < max\_cols; l++) {**

**/\***

**\* (i, j) 节点中位置 (k, l) 的元素对应了原矩阵中**

**\* 位置 (i \* max\_rows\_a + k, j \* max\_cols\_b + l) 的元素**

**\*/**

**temp[k \* max\_cols + l] = (i \* max\_rows + k < row && j \* max\_cols + l < col) ?**

**buffer[(i \* max\_rows + k) \* col + j \* max\_cols + l] : 0;**

**}**

**/\***

**\* i 和 j 不全是 0 表明目标节点不是 0 号节点，此时需要使用 MPI 完成传送**

**\* 否则只需要由 0 号节点自身完成拷贝即可**

**\*/**

**if (i || j) {**

**/\***

**\* 对 A 矩阵的块 (i, j) 初始化时需要循环左移 i 列**

**\* 对 B 矩阵的块 (i, j) 初始化是需要循环上移 j 列**

**\*/**

**MPI\_Send(temp, total, MPI\_DOUBLE, tag == SCATTER\_A\_TAG ?**

**shift\_left(i, j, i) : shift\_up(i, j, j),**

**tag, MPI\_COMM\_WORLD);**

**}**

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

**buf[k] = temp[k];**

**}**

**}**

**free(temp);**

**}**

**else {**

**/\***

**\* 非 0 节点不参与分发矩阵到其它节点**

**\* 只需要接收来自 0 号节点的矩阵即可**

**\*/**

**MPI\_Status status;**

**MPI\_Recv(buf, total, MPI\_DOUBLE, 0, tag, MPI\_COMM\_WORLD, &status);**

**}**

**}**

**/\***

**\* 佳能算法的核心计算部分**

**\* 理论上可以使用 inline 关键字标记以使编译器内联此函数，提高性能**

**\* 然而在 mpicc 中开启 -O2 优化后 inline 关键字会诡异的导致 Runtime Error**

**\* 为使程序有更好的可读性，仍封装成函数**

**\*/**

**void cannon(double\* block\_a, double\* block\_b, double\* block\_c**

**, double\* buffer\_a, double\* buffer\_b, double\* buffer\_c) {**

**/\***

**\* 初始化存储当前节点计算结果的块为 0**

**\* 在后续过程中不断对 block\_c 进行累加**

**\*/**

**for (int i = 0; i < max\_rows\_a \* max\_cols\_b; i++) {**

**block\_c[i] = 0;**

**}**

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

**/\***

**\* 此处使用了类似 Copy On Write 的思想，维护了 block 和 buffer：**

**\* 其中的 block 是只读的，即传输过程和计算过程中都不会对其进行更新**

**\* 其中的 buffer 是只写的，即传输过程和计算过程中都不会对其进行读取**

**\* 上述性质允许了同时进行传输和计算，效率得以进一步提高**

**\*/**

**MPI\_Status status[4];**

**MPI\_Request request[4];**

**/\***

**\* 接下来四个函数调用对应了两个处理：**

**\* 将 A 矩阵中正在处理的块 (my\_id / rooted\_proc, my\_id % rooted\_proc) 循环左移一个单位**

**\* 将 B 矩阵中正在处理的块 (my\_id / rooted\_proc, my\_id % rooted\_proc) 循环上移一个单位**

**\* 开启传输后，在当前节点上串行完成矩阵乘法计算**

**\*/**

**MPI\_Irecv(buffer\_a, max\_rows\_a \* max\_cols\_a, MPI\_DOUBLE,**

**shift\_left(my\_id / rooted\_proc, my\_id % rooted\_proc, -1),**

**SCATTER\_A\_TAG, MPI\_COMM\_WORLD, &request[0]);**

**MPI\_Irecv(buffer\_b, max\_rows\_b \* max\_cols\_b, MPI\_DOUBLE,**

**shift\_up(my\_id / rooted\_proc, my\_id % rooted\_proc, -1),**

**SCATTER\_B\_TAG, MPI\_COMM\_WORLD, &request[1]);**

**MPI\_Isend(block\_a, max\_rows\_a \* max\_cols\_a, MPI\_DOUBLE,**

**shift\_left(my\_id / rooted\_proc, my\_id % rooted\_proc, 1),**

**SCATTER\_A\_TAG, MPI\_COMM\_WORLD, &request[2]);**

**MPI\_Isend(block\_b, max\_rows\_b \* max\_cols\_b, MPI\_DOUBLE,**

**shift\_up(my\_id / rooted\_proc, my\_id % rooted\_proc, 1),**

**SCATTER\_B\_TAG, MPI\_COMM\_WORLD, &request[3]);**

**for (int j = 0; j < max\_rows\_a; j++) for (int k = 0; k < max\_cols\_b; k++) {**

**/\***

**\* 采用 temp 缓存，一方面保持 buffer 的只写性**

**\* 另一方面减少访存开销以提升效率**

**\*/**

**double temp = 0;**

**for (int l = 0; l < max\_cols\_a; l++) {**

**temp += block\_a[j \* max\_cols\_a + l] \* block\_b[l \* max\_cols\_b + k];**

**}**

**buffer\_c[j \* max\_cols\_b + k] = temp;**

**}**

**/\***

**\* 传输和计算结束，开始交换 block 和 buffer 的角色**

**\* 对于结果 C 的交换不依赖于 A 和 B，因此可在同步前进行以提升效率**

**\* 对 A 和 B 的交换需要等待传输和计算完成，因此需要进行一次同步**

**\* 对 A 和 B 的交换无需遍历进行赋值，只需要交换两个指针即可**

**\*/**

**for (int j = 0; j < max\_rows\_a \* max\_cols\_b; j++) {**

**block\_c[j] += buffer\_c[j];**

**}**

**MPI\_Waitall(4, request, status);**

**swap(&block\_a, &buffer\_a), swap(&block\_b, &buffer\_b);**

**}**

**}**

**/\***

**\* 计算结束，整合结果得到目标矩阵**

**\* 理论上可以使用 inline 关键字标记以使编译器内联此函数，提高性能**

**\* 然而在 mpicc 中开启 -O2 优化后 inline 关键字会诡异的导致 Runtime Error**

**\* 为使程序有更好的可读性，仍封装成函数**

**\*/**

**void gather\_matrix(double\* source\_c, double\* block\_c, double\* buffer\_c) {**

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

**for (int i = 0; i < rooted\_proc; i++) for (int j = 0; j < rooted\_proc; j++) {**

**/\***

**\* 遍历 i 和 j 并将其计算结果汇总到 0 号节点**

**\* i 和 j 不全是 0 表明目标节点不是 0 号节点，此时需要使用 MPI 完成传送**

**\* 否则只需要由 0 号节点自身完成拷贝即可**

**\*/**

**if (i || j){**

**MPI\_Status status;**

**MPI\_Recv(buffer\_c, max\_rows\_a \* max\_cols\_b, MPI\_DOUBLE,**

**i \* rooted\_proc + j, SHIFT\_TAG, MPI\_COMM\_WORLD, &status);**

**}**

**else for (int k = 0; k < max\_rows\_a \* max\_cols\_b; k++) {**

**buffer\_c[k] = block\_c[k];**

**}**

**/\***

**\* 整合 (i, j) 节点对应的数据块到合适的位置**

**\* 对于不能整除时出现的空间冗余，忽略多余的位置**

**\*/**

**for (int k = 0; k < max\_rows\_a; k++) for (int l = 0; l < max\_cols\_b; l++) {**

**if (i \* max\_rows\_a + k < n && j \* max\_cols\_b + l < p) {**

**/\***

**\* (i, j) 节点中位置 (k, l) 的元素对应了原矩阵中**

**\* 位置 (i \* max\_rows\_a + k, j \* max\_cols\_b + l) 的元素**

**\*/**

**source\_c[(i \* max\_rows\_a + k) \* p + j \* max\_cols\_b + l] =**

**buffer\_c[k \* max\_cols\_b + l];**

**}**

**}**

**}**

**}**

**else {**

**/\***

**\* 非 0 节点不参与处理来自其它矩阵的节点**

**\* 只需要将自身的处理结果发送到 0 号节点即可**

**\*/**

**MPI\_Send(block\_c, max\_rows\_a \* max\_cols\_b, MPI\_DOUBLE,**

**0, SHIFT\_TAG, MPI\_COMM\_WORLD);**

**}**

**}**

**/\***

**\* 读文件过程，返回读取到的矩阵**

**\* 读取的矩阵的存储空间在此函数中完成申请**

**\* 需要外部在合适的时候释放申请的这部分内存**

**\* 通过 row 指针和 col 指针的方式，间接返回矩阵的大小信息**

**\*/**

**double\* read\_file(const char\* filename, int\* row, int\* col) {**

**/\***

**\* 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);**

**\*row = ((int\*)f\_stream)[0], \*col = ((int\*)f\_stream)[1];**

**/\***

**\* 处理异常情况：**

**\* 矩阵的长宽必须是正数**

**\*/**

**if (\*row <= 0 || \*col <= 0) {**

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

**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) \* \*row \* \*col)) {**

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

**exit(-1);**

**}**

**/\***

**\* 为读取出来的矩阵申请空间**

**\* 循环对读取结果进行赋值，返回结果**

**\*/**

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

**double\* matrix = (double\*)malloc(sizeof(double) \* \*row \* \*col);**

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

**for (int i = 0; i < \*row; i++) for (int j = 0; j < \*col; j++) {**

**matrix[i \* \*col + j] = matrix\_addr[i \* \*col + j];**

**}**

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

**return matrix;**

**}**

**/\***

**\* 写文件过程，输出传入的矩阵**

**\* row 和 col 分别对应矩阵的行数和列数**

**\*/**

**void write\_file(const char\* filename, const int row, const int col, const double\* matrix) {**

**/\***

**\* 申请存储空间，并记录矩阵信息**

**\* 存储到文件的矩阵，最开始的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) \* row \* col);**

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

**((int\*)buffer)[0] = row, ((int\*)buffer)[1] = col;**

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

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

**for (int i = 0; i < row; i++, putchar('\n')) for (int j = 0; j < col; j++) {**

**/\***

**\* 循环遍历传入的矩阵**

**\* 输出矩阵信息并写入缓存**

**\*/**

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

**ptr[i \* col + j] = matrix[i \* col + j];**

**}**

**FILE\* file\_ptr;**

**/\***

**\* 开文件并特判**

**\* 文件不存在直接报错并退出，而不是只报错不退出**

**\* 否则会因空指针出现 Runtime 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);**

**free(buffer), fclose(file\_ptr);**

**}**

**/\***

**\* 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 Runtime 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[]) {**

**/\***

**\* 初始化 MPI 环境并获取节点总数，自身 id**

**\* 在 cannon 算法中，节点总数必须为完全平方数**

**\*/**

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

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

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

**rooted\_proc = (int)sqrt(num\_proc);**

**if (num\_proc != rooted\_proc \* rooted\_proc) {**

**printf("Processor number must be a square!\n");**

**exit(-1);**

**}**

**double\* source\_a = NULL; double\* source\_b = NULL; double\* source\_c = NULL;**

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

**/\***

**\* 0 号节点读矩阵文件，并为计算结果申请空间**

**\* 读矩阵的同时完成 0 号节点中，对矩阵尺寸的更新**

**\*/**

**source\_a = read\_file("matrix\_a.stdin", &n, &m);**

**source\_b = read\_file("matrix\_b.stdin", &m, &p);**

**source\_c = (double\*)malloc(sizeof(double) \* n \* p);**

**}**

**/\***

**\* MPI\_Bcast广播发送消息，将矩阵的大小信息广播到 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**

**\* 各个节点根据大小信息计算块的大小，并对各个块申请空间**

**\* 如果不能整除，则实际大小为向上取整后的大小**

**\*/**

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

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

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

**max\_rows\_a = (n + rooted\_proc - 1) / rooted\_proc, max\_cols\_a = (m + rooted\_proc - 1) / rooted\_proc;**

**max\_rows\_b = (m + rooted\_proc - 1) / rooted\_proc, max\_cols\_b = (p + rooted\_proc - 1) / rooted\_proc;**

**double\* block\_a = (double\*)malloc(sizeof(double) \* max\_rows\_a \* max\_cols\_a);**

**double\* buffer\_a = (double\*)malloc(sizeof(double) \* max\_rows\_a \* max\_cols\_a);**

**double\* block\_b = (double\*)malloc(sizeof(double) \* max\_rows\_b \* max\_cols\_b);**

**double\* buffer\_b = (double\*)malloc(sizeof(double) \* max\_rows\_b \* max\_cols\_b);**

**double\* block\_c = (double\*)malloc(sizeof(double) \* max\_rows\_a \* max\_cols\_b);**

**double\* buffer\_c = (double\*)malloc(sizeof(double) \* max\_rows\_a \* max\_cols\_b);**

**/\***

**\* 空间申请完成，分别分发 A 矩阵和 B 矩阵到各个节点**

**\* 分发完成即表明 IO 完成，开始计时**

**\*/**

**scatter\_matrix(block\_a, n, m, SCATTER\_A\_TAG, source\_a);**

**scatter\_matrix(block\_b, m, p, SCATTER\_B\_TAG, source\_b);**

**MPI\_Barrier(MPI\_COMM\_WORLD);**

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

**/\***

**\* 调用 cannon 算法核心核心计算部分完成计算**

**\* 完成计算后同步一次**

**\* 合并计算结果，得到目标矩阵**

**\* 完成计算后同步一次**

**\* 此时计算完成而 IO 即将开始，计时结束**

**\*/**

**cannon(block\_a, block\_b, block\_c, buffer\_a, buffer\_b, buffer\_c);**

**MPI\_Barrier(MPI\_COMM\_WORLD);**

**gather\_matrix(source\_c, block\_c, buffer\_c);**

**MPI\_Barrier(MPI\_COMM\_WORLD);**

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

**/\***

**\* 开始输出，只有 0 号节点需要输出和释放三个 source 对应的空间**

**\* 全部节点都要释放其 block 和 buffer 对应的空间**

**\*/**

**if (my\_id == 0) {**

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

**write\_file("result.stdout", n, p, source\_c);**

**free(source\_a), free(source\_b), free(source\_c);**

**}**

**free(buffer\_a), free(buffer\_b), free(buffer\_c);**

**free(block\_a), free(block\_b), free(block\_c);**

**MPI\_Finalize(); return 0;**

**}**

## 算法的设计方法分析

* 划分：根据矩阵乘法中累加的可分离性，原矩阵分解成个矩阵块，每个节点只进行矩阵块的乘法。
* 通讯：分发矩阵和整合矩阵的阶段，采用的全局通讯，计算阶段采用结构化通讯。
* 组合：每个节点负责一个矩阵块的串行计算，同时负责矩阵块之间的传递和通讯。
* 映射：任务的划分个数等于处理器个数，组合任务的同时完成了处理器映射。

## 算法的并行模式分析

* 主从并行模式：一个节点作为，其它节点作为。在算法分发矩阵和整合矩阵的阶段，采用了这种并行模式，号节点作为，其它节点作为。
* 流水线并行模式：数据流通过一串计算任务，前一计算任务的输出是下一个计算任务的输入。在算法的计算阶段采用了这种并行模式，其正确性的依据在于：，可依次进行，矩阵乘法中累加的可分离性。

## 算法时间复杂度分析

* 读入部分时间复杂度显然为：
* 广播矩阵大小到各个节点的时间复杂度为：
* 各个节点申请空间的时间复杂度显然为：
* 分发矩阵的过程中，非节点接收了一个的矩阵，时间复杂度为：
* 分发矩阵的过程中，号节点发送了个的矩阵，时间复杂度为：
* 分发过程进行了两次，总时间复杂度为：
* 计算过程每一轮循环中，节点发送了两个的矩阵，接收了两个的矩阵，时间复杂度为：
* 计算过程每一轮循环中，节点完成了两个的矩阵的乘法，时间复杂度为：
* 计算过程总共进行了轮循环，总时间复杂度为：
* 合并矩阵的过程中，非节点发送了一个的矩阵，时间复杂度为：
* 合并矩阵的过程中，号节点接收了个的矩阵，时间复杂度为：
* 输出部分时间复杂度显然为：
* 综上所述，此算法的总时间复杂度为：

## 算法空间复杂度分析

* 号节点存储了两个原矩阵和目标矩阵，空间复杂度为：
* 每个节点存储了三个矩阵块及其缓存，每个矩阵块大小为，空间复杂度为：
* 综上所述，此算法的总空间复杂度为：

## 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+用两种方式计算+输出/比对结果

**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\_cannon.c -o mult\_cannon ' + ' '.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()**

## 实验结果及

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**

**6 10 8**

**---- matrix\_a.stdin: 6 \* 10 Matrix -----**

**0.7635 0.6909 0.8034 0.8465 0.4129 0.5978 0.8274 0.4947 0.7300 0.1576**

**0.0314 0.8127 0.2495 0.0968 0.3159 0.7363 0.2386 0.5067 0.0226 0.7114**

**0.1007 0.8226 0.6535 0.8245 0.1789 0.1244 0.5791 0.9711 0.0553 0.2145**

**0.0744 0.8188 0.9053 0.8779 0.6653 0.3182 0.4756 0.4927 0.8129 0.2056**

**0.6503 0.8444 0.0184 0.8998 0.9412 0.3343 0.6360 0.1798 0.8410 0.6586**

**0.8912 0.9417 0.4812 0.5447 0.7662 0.6601 0.6690 0.3453 0.6312 0.7244**

**---- matrix\_b.stdin: 10 \* 8 Matrix -----**

**0.6880 0.0921 0.4102 0.2994 0.0555 0.1371 0.2750 0.3320**

**0.3148 0.0497 0.7152 0.7599 0.0069 0.4702 0.2659 0.8244**

**0.8272 0.6891 0.0417 0.6258 0.3168 0.9443 0.3837 0.2846**

**0.0893 0.8471 0.7940 0.3161 0.9245 0.8116 0.6253 0.6125**

**0.9037 0.0354 0.9119 0.9591 0.1725 0.1868 0.2912 0.4873**

**0.2365 0.0064 0.2472 0.2434 0.4765 0.5131 0.0678 0.3037**

**0.2022 0.1095 0.9295 0.5190 0.0539 0.3132 0.8036 0.1432**

**0.1603 0.5976 0.4592 0.0848 0.4092 0.0845 0.6973 0.3129**

**0.1199 0.6092 0.2720 0.2924 0.7960 0.5631 0.7798 0.0325**

**0.5695 0.0270 0.2759 0.0461 0.5401 0.3437 0.3498 0.7423**

**Serial algorithm: multiply a 6 x 10 with a 10 x 8, use 0.00(s)**

**---- standard.stdout: 6 \* 8 Matrix -----**

**2.4213 2.2290 3.2755 2.7575 2.3535 3.0252 3.0262 2.3668**

**1.4894 0.6750 1.8084 1.5021 1.2037 1.5121 1.3345 1.9099**

**1.5351 1.8893 2.5638 1.9356 1.6591 2.1523 2.3338 2.0752**

**2.2025 2.2880 2.9824 2.7399 2.3600 2.9692 2.7813 2.3172**

**2.3717 1.6194 3.6109 2.7379 2.3342 2.4955 2.7923 2.6921**

**2.8836 1.6369 3.5055 2.9458 2.2297 2.7864 2.8125 2.8788**

**Submitted batch job 1768883**

**---- result.stdout: 6 \* 8 Matrix -----**

**2.4213 2.2290 3.2755 2.7575 2.3535 3.0252 3.0262 2.3668**

**1.4894 0.6750 1.8084 1.5021 1.2037 1.5121 1.3345 1.9099**

**1.5351 1.8893 2.5638 1.9356 1.6591 2.1523 2.3338 2.0752**

**2.2025 2.2880 2.9824 2.7399 2.3600 2.9692 2.7813 2.3172**

**2.3717 1.6194 3.6109 2.7379 2.3342 2.4955 2.7923 2.6921**

**2.8836 1.6369 3.5055 2.9458 2.2297 2.7864 2.8125 2.8788**

**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**

**1600 1600 1600**

**Serial algorithm: multiply a 1600 x 1600 with a 1600 x 1600, use 15.20(s)**

**Submitted batch job 1768884**

**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 homework4]$ cat run\_time.stdout**

**Total time is 0.507468**

## 基于性能分析软件进行分析

在集群上安装的包如下，对部分版本进行了调整：

**[pc19373354@ln01 packages]$ pwd**

**/gs/home/pc19373354/packages**

**[pc19373354@ln01 packages]$ tree -L 1**

**.**

**├── cubelib-4.4**

**├── cubelib-4.4.tar.gz**

**├── cubew-4.4**

**├── cubew-4.4.tar.gz**

**├── opari2-2.0.3**

**├── opari2-2.0.3.tar.gz**

**├── otf2-2.1.1**

**├── otf2-2.1.1.tar.gz**

**├── scorep-4.0**

**└── scorep-4.0.tar.gz**

**5 directories, 5 files**

安装位置为下的，具体如下：

**[pc19373354@ln01 .local]$ pwd**

**/gs/home/pc19373354/.local**

**[pc19373354@ln01 .local]$ tree -L 2**

**.**

**├── bin**

**│   ├── cube3to4**

**│   ├── cube4to3**

**│   ├── cube\_calltree**

**│   ├── cube\_canonize**

**│   ├── cube\_clean**

**│   ├── cube\_cmp**

**│   ├── cube\_commoncalltree**

**│   ├── cube\_cut**

**│   ├── cube\_derive**

**│   ├── cube\_diff**

**│   ├── cube\_dump**

**│   ├── cube\_exclusify**

**│   ├── cube\_inclusify**

**│   ├── cube\_info**

**│   ├── cube\_is\_empty**

**│   ├── cubelib-config**

**│   ├── cube\_mean**

**│   ├── cube\_merge**

**│   ├── cube\_nodeview**

**│   ├── cube\_part**

**│   ├── cube\_rank**

**│   ├── cube\_regioninfo**

**│   ├── cube\_remap2**

**│   ├── cube\_sanity**

**│   ├── cube\_score**

**│   ├── cube\_server**

**│   ├── cube\_stat**

**│   ├── cube\_test**

**│   ├── cube\_topoassist**

**│   ├── cubew-config**

**│   ├── opari2**

**│   ├── opari2-config**

**│   ├── otf2-config**

**│   ├── otf2-estimator**

**│   ├── otf2-marker**

**│   ├── otf2-print**

**│   ├── otf2-snapshots**

**│   ├── otf2-template**

**│   ├── scorep**

**│   ├── scorep-backend-info**

**│   ├── scorep-config**

**│   ├── scorep-g++ -> scorep-wrapper**

**│   ├── scorep-gcc -> scorep-wrapper**

**│   ├── scorep-gfortran -> scorep-wrapper**

**│   ├── scorep-info**

**│   ├── scorep-mpicc -> scorep-wrapper**

**│   ├── scorep-mpicxx -> scorep-wrapper**

**│   ├── scorep-mpif77 -> scorep-wrapper**

**│   ├── scorep-mpif90 -> scorep-wrapper**

**│   ├── scorep-online-access-registry**

**│   ├── scorep-score**

**│   ├── scorep-wrapper**

**│   └── tau2cube**

**├── include**

**│   ├── cubelib**

**│   ├── cubew**

**│   ├── opari2**

**│   ├── otf2**

**│   └── scorep**

**├── lib**

**│   ├── libcube4.a**

**│   ├── libcube4.la**

**│   ├── libcube4.so -> libcube4.so.7.1.0**

**│   ├── libcube4.so.7 -> libcube4.so.7.1.0**

**│   ├── libcube4.so.7.1.0**

**│   ├── libcube4w.a**

**│   ├── libcube4w.la**

**│   ├── libcube.tools.common.a**

**│   ├── libcube.tools.common.la**

**│   ├── libcube.tools.common.so -> libcube.tools.common.so.0.0.0**

**│   ├── libcube.tools.common.so.0 -> libcube.tools.common.so.0.0.0**

**│   ├── libcube.tools.common.so.0.0.0**

**│   ├── libcube.tools.library.a**

**│   ├── libcube.tools.library.la**

**│   ├── libcube.tools.library.so -> libcube.tools.library.so.0.0.0**

**│   ├── libcube.tools.library.so.0 -> libcube.tools.library.so.0.0.0**

**│   ├── libcube.tools.library.so.0.0.0**

**│   ├── libcube.tools.remapper.a**

**│   ├── libcube.tools.remapper.la**

**│   ├── libcube.tools.remapper.so -> libcube.tools.remapper.so.0.0.0**

**│   ├── libcube.tools.remapper.so.0 -> libcube.tools.remapper.so.0.0.0**

**│   ├── libcube.tools.remapper.so.0.0.0**

**│   ├── libotf2.a**

**│   ├── libotf2.la**

**│   ├── libscorep\_adapter\_compiler\_event.a**

**│   ├── libscorep\_adapter\_compiler\_event.la**

**│   ├── libscorep\_adapter\_compiler\_mgmt.a**

**│   ├── libscorep\_adapter\_compiler\_mgmt.la**

**│   ├── libscorep\_adapter\_memory\_event\_cxx.a**

**│   ├── libscorep\_adapter\_memory\_event\_cxx\_L32.a**

**│   ├── libscorep\_adapter\_memory\_event\_cxx\_L32.la**

**│   ├── libscorep\_adapter\_memory\_event\_cxx\_L64.a**

**│   ├── libscorep\_adapter\_memory\_event\_cxx\_L64.la**

**│   ├── libscorep\_adapter\_memory\_event\_cxx.la**

**│   ├── libscorep\_adapter\_memory\_event\_libc11.a**

**│   ├── libscorep\_adapter\_memory\_event\_libc11.la**

**│   ├── libscorep\_adapter\_memory\_event\_libc.a**

**│   ├── libscorep\_adapter\_memory\_event\_libc.la**

**│   ├── libscorep\_adapter\_memory\_event\_pgCC.a**

**│   ├── libscorep\_adapter\_memory\_event\_pgCC\_L32.a**

**│   ├── libscorep\_adapter\_memory\_event\_pgCC\_L32.la**

**│   ├── libscorep\_adapter\_memory\_event\_pgCC\_L64.a**

**│   ├── libscorep\_adapter\_memory\_event\_pgCC\_L64.la**

**│   ├── libscorep\_adapter\_memory\_event\_pgCC.la**

**│   ├── libscorep\_adapter\_memory\_mgmt.a**

**│   ├── libscorep\_adapter\_memory\_mgmt.la**

**│   ├── libscorep\_adapter\_mpi\_event.a**

**│   ├── libscorep\_adapter\_mpi\_event.la**

**│   ├── libscorep\_adapter\_mpi\_mgmt.a**

**│   ├── libscorep\_adapter\_mpi\_mgmt.la**

**│   ├── libscorep\_adapter\_opari2\_mgmt.a**

**│   ├── libscorep\_adapter\_opari2\_mgmt.la**

**│   ├── libscorep\_adapter\_opari2\_openmp\_event.a**

**│   ├── libscorep\_adapter\_opari2\_openmp\_event.la**

**│   ├── libscorep\_adapter\_opari2\_openmp\_mgmt.a**

**│   ├── libscorep\_adapter\_opari2\_openmp\_mgmt.la**

**│   ├── libscorep\_adapter\_opari2\_user\_event.a**

**│   ├── libscorep\_adapter\_opari2\_user\_event.la**

**│   ├── libscorep\_adapter\_opari2\_user\_mgmt.a**

**│   ├── libscorep\_adapter\_opari2\_user\_mgmt.la**

**│   ├── libscorep\_adapter\_pthread\_event.a**

**│   ├── libscorep\_adapter\_pthread\_event.la**

**│   ├── libscorep\_adapter\_pthread\_mgmt.a**

**│   ├── libscorep\_adapter\_pthread\_mgmt.la**

**│   ├── libscorep\_adapter\_user\_event.a**

**│   ├── libscorep\_adapter\_user\_event.la**

**│   ├── libscorep\_adapter\_user\_mgmt.a**

**│   ├── libscorep\_adapter\_user\_mgmt.la**

**│   ├── libscorep\_adapter\_utils.a**

**│   ├── libscorep\_adapter\_utils.la**

**│   ├── libscorep\_alloc\_metric.a**

**│   ├── libscorep\_alloc\_metric.la**

**│   ├── libscorep\_estimator.a**

**│   ├── libscorep\_estimator.la**

**│   ├── libscorep\_estimator.so -> libscorep\_estimator.so.0.0.0**

**│   ├── libscorep\_estimator.so.0 -> libscorep\_estimator.so.0.0.0**

**│   ├── libscorep\_estimator.so.0.0.0**

**│   ├── libscorep\_measurement.a**

**│   ├── libscorep\_measurement.la**

**│   ├── libscorep\_mpp\_mockup.a**

**│   ├── libscorep\_mpp\_mockup.la**

**│   ├── libscorep\_mpp\_mpi.a**

**│   ├── libscorep\_mpp\_mpi.la**

**│   ├── libscorep\_mutex\_mockup.a**

**│   ├── libscorep\_mutex\_mockup.la**

**│   ├── libscorep\_mutex\_omp.a**

**│   ├── libscorep\_mutex\_omp.la**

**│   ├── libscorep\_mutex\_pthread.a**

**│   ├── libscorep\_mutex\_pthread.la**

**│   ├── libscorep\_mutex\_pthread\_spinlock.a**

**│   ├── libscorep\_mutex\_pthread\_spinlock.la**

**│   ├── libscorep\_mutex\_pthread\_wrap.a**

**│   ├── libscorep\_mutex\_pthread\_wrap.la**

**│   ├── libscorep\_online\_access\_mockup.a**

**│   ├── libscorep\_online\_access\_mockup.la**

**│   ├── libscorep\_online\_access\_mpi\_mockup.a**

**│   ├── libscorep\_online\_access\_mpi\_mockup.la**

**│   ├── libscorep\_online\_access\_mpp\_mpi.a**

**│   ├── libscorep\_online\_access\_mpp\_mpi.la**

**│   ├── libscorep\_online\_access\_spp.a**

**│   ├── libscorep\_online\_access\_spp.la**

**│   ├── libscorep\_thread\_create\_wait\_pthread.a**

**│   ├── libscorep\_thread\_create\_wait\_pthread.la**

**│   ├── libscorep\_thread\_fork\_join\_omp.a**

**│   ├── libscorep\_thread\_fork\_join\_omp.la**

**│   ├── libscorep\_thread\_mockup.a**

**│   ├── libscorep\_thread\_mockup.la**

**│   └── scorep**

**├── libexec**

**│   ├── pomp2-parse-init-regions.awk**

**│   └── scorep**

**└── share**

**├── cubelib**

**├── cubew**

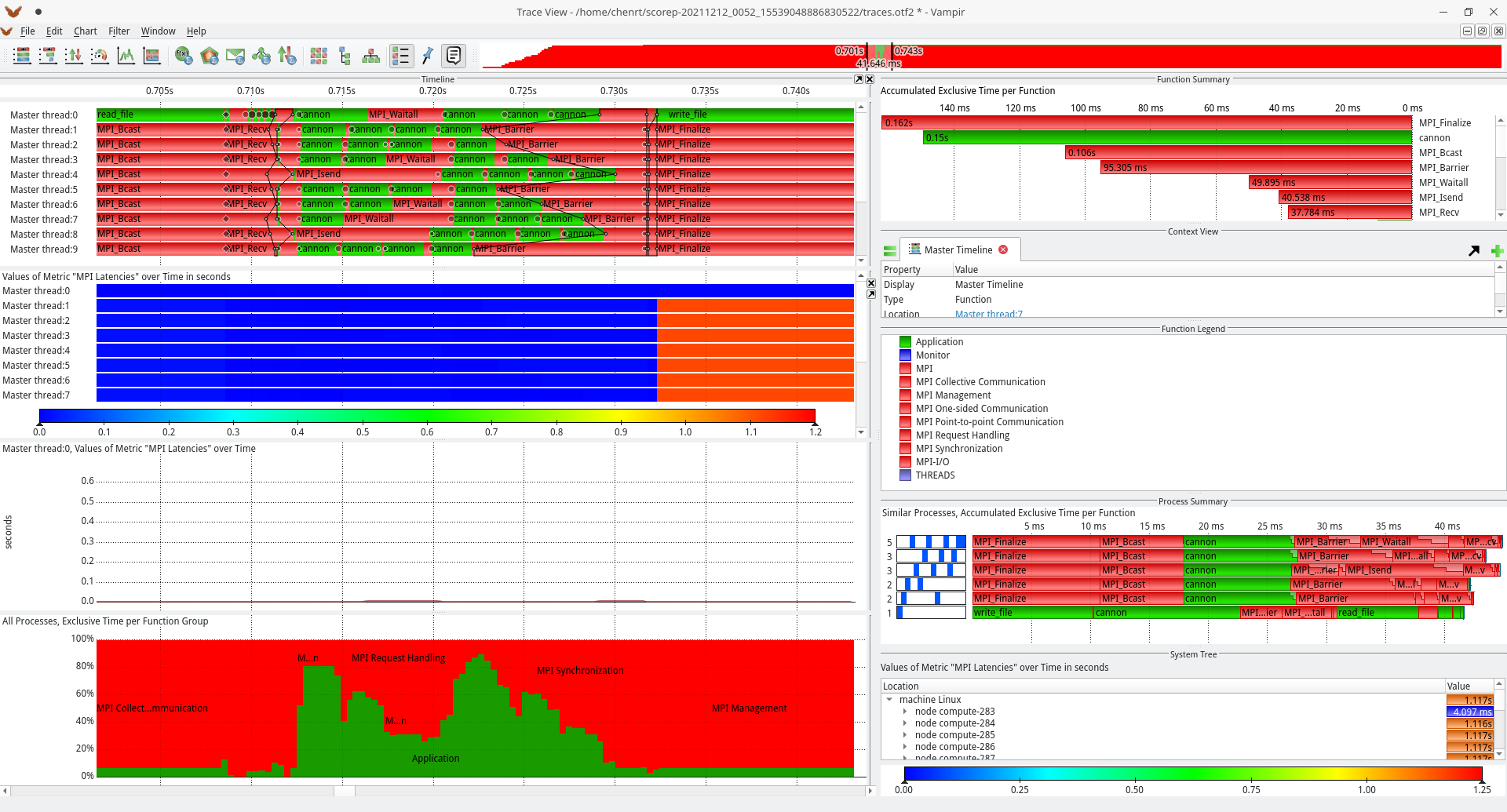
**├── doc**

**├── opari2**

**├── otf2**

**└── scorep**

**18 directories, 161 files**

在本地()使用查看分析结果，如下：