**并行计算第二次作业**

## 程序流程图

## 注释后的实验源程序

**#include <mpi.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**/\***

**\* 定义不同的 tag**

**\* 用以区分进程通信时的消息**

**\*/**

**#define MULTI\_LEN 600**

**#define MULTI\_TYPE 300**

**#define OUTPUT\_TYPE 200**

**#define ALLTOONE\_TYPE 100**

**/\* 输出错误信息 \*/**

**void merror(char\* s) {**

**printf("%s\n", s);**

**exit(1);**

**}**

**/\* 交换两个变量 \*/**

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

**int t = \*x;**

**\*x = \*y, \*y = t;**

**}**

**/\* 串行快速排序算法 \*/**

**void quick\_sort(int data[], int l, int r) {**

**if (l > r) return;**

**int mid = (l + r) / 2;**

**/\***

**\* 对源程序进行了修改**

**\* 通过三点取中的方式避免被基本有序的数据卡**

**\*/**

**if (data[l] > data[r]) swap(data + l, data + r);**

**if (data[r] < data[mid]) swap(data + r, data + mid);**

**if (data[l] < data[mid]) swap(data + l, data + mid);**

**int i = l, j = r, temp = data[l];**

**/\* 完成对基准元素的归位 \*/**

**while (i - j) {**

**while (data[j] >= temp && i < j) j--;**

**while (data[i] <= temp && i < j) i++;**

**if (i < j) swap(data + i, data + j);**

**}**

**data[l] = data[i], data[i] = temp, mid = j = i;**

**/\***

**\* 将直接在基准元素两侧递归改成两侧去重后递归**

**\* 不进行此修改时会被全部相同的数据卡**

**\*/**

**while (i >= l && data[i] == data[mid]) i--;**

**while (j <= r && data[j] == data[mid]) j++;**

**/\* 在基准元素两侧递归快速排序 \*/**

**quick\_sort(data, l, i);**

**quick\_sort(data, j, r);**

**}**

**/\* 合并两个有序数组 \*/**

**void merge(const int src[], int dst[], int l, int mid, int r) {**

**/\***

**\* 修改了输入格式并重命名了变量**

**\* 原来的 s1 对应现在的 mid**

**\* 原来的 s1 + s2 对应现在的 r**

**\* 额外传入参数 l 使函数更具有一般性**

**\*/**

**int index = 0, l\_ptr = l, r\_ptr = mid;**

**while (l\_ptr < mid && r\_ptr < r) {**

**if (src[l\_ptr] > src[r\_ptr]) dst[index++] = src[r\_ptr++];**

**else dst[index++] = src[l\_ptr++];**

**}**

**/\***

**\* 将原本的 for 循环按照 l\_ptr < mid && r\_ptr < r拆开**

**\* 削弱了循环内部分支的复杂程度**

**\* 有利于编译器产生更高效的汇编代码**

**\*/**

**while (l\_ptr < mid) dst[index++] = src[l\_ptr++];**

**while (r\_ptr < r) dst[index++] = src[r\_ptr++];**

**}**

**/\* 串行多路归并算法递归部分 \*/**

**void recursion\_merge(int src[], int buf[], int index[], int l, int r) {**

**if (r - l < 1) return;**

**int mid = (l + r) / 2;**

**/\***

**\* 分别递归两边，得到两个有序子数组**

**\* 再合并得到的两个有序子数组**

**\*/**

**recursion\_merge(src, buf, index, l, mid);**

**recursion\_merge(src, buf, index, mid + 1, r);**

**/\***

**\* 多路归并的关键在于“分治”**

**\* 对于 [1], [2], [3], [4], [5], [6], [7], [8] 这八个数组，合并的方式是：**

**\* [1], [2], [3], [4], [5], [6], [7], [8]**

**\* [1, 2], [3, 4], [5, 6], [7, 8]**

**\* [1, 2, 3, 4], [5, 6, 7, 8]**

**\* [1, 2, 3, 4, 5, 6, 7, 8]**

**\* 而不是：**

**\* [1, 2], [3], [4], [5], [6], [7], [8]**

**\* [1, 2, 3], [4], [5], [6], [7], [8]**

**\* [1, 2, 3, 4], [5], [6], [7], [8]**

**\* [1, 2, 3, 4, 5], [6], [7], [8]**

**\* [1, 2, 3, 4, 5, 6], [7], [8]**

**\* [1, 2, 3, 4, 5, 6, 7], [8]**

**\* [1, 2, 3, 4, 5, 6, 7, 8]**

**\* 前者的复杂度是 (n / p) \* p \* log(p) = n \* log(p) 的**

**\* 后者的复杂度是 (n / p) \* p \* p = n \* p 的**

**\*/**

**merge(src, buf, index[l], index[mid + 1], index[r + 1]);**

**/\***

**\* 不采用原实现中的“滚动数组”**

**\* 两个数组一个只作为原数组使用，一个只作为 buffer 使用**

**\* 不会出现原实现中需要根据奇偶性判断最终结果在哪一个数组的情况**

**\* 因此在完成一次归并后，需要更新 src 数组对应的位置**

**\*/**

**for (int i = index[l]; i < index[r + 1]; i++) {**

**src[i] = buf[i - index[l]];**

**}**

**/\***

**\* 此外，多路归并算法还可以通过堆实现，亦有 n \* log(p) 的复杂度**

**\* 相比分治实现，堆实现有更小的函数调用开销，但空间连续性不如分治实现**

**\* 在不同的机器上运行，两种方式可能有不同的效果**

**\*/**

**}**

**/\* 串行多路归并算法递归前部分 \*/**

**void multi\_merge(int src[], int buf[], const int length[], int total) {**

**int\* index = (int\*)malloc((total + 1) \* sizeof(int));**

**index[0] = 0;**

**/\***

**\* 输入的 length 意义是每一段子数组的长度**

**\* 为了在递归部分中快速得到每一段子数组的下标信息，需对 length 数组求和**

**\* 本质上是前缀和优化，实现 O(1) 得到下标**

**\*/**

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

**index[i] = index[i - 1] + length[i - 1];**

**}**

**/\***

**\* 原实现本质上还是分治实现，不过其尾递归实现更像是“披着递归皮的循环”**

**\* 且其中存在较多特殊处理，如对 length 数组压缩和清零，对区间数的奇偶特判等**

**\* 考虑到其原实现不够清晰优雅，对此部分进行了重写**

**\*/**

**recursion\_merge(src, buf, index, 0, total - 1);**

**free(index);**

**}**

**/\* 并行正则采样排序的主体 \*/**

**void psrs\_main(int data\_size) {**

**int id, total, mutex;**

**MPI\_Status status[32 \* 32 \* 2];**

**MPI\_Request request[32 \* 32 \* 2];**

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

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

**/\***

**\* 申请空间并完成初始化 split 和 length**

**\* 此处采用一次性 malloc 出 data 和 buffer 的方式**

**\* 在归并时避免了重复 malloc 出 buffer 造成的性能下降**

**\***

**\* split 是分段的数量**

**\* length 是此处理器处理的数据量**

**\* data 是长度为 data\_size 的数组，存原始数据**

**\* buffer 是长度为 data\_size 的数组，作为缓冲区**

**\*/**

**int split = total - 1;**

**int length = data\_size / total;**

**int\* data = (int\*)malloc(data\_size \* sizeof(int) \* 2);**

**if (data == 0) merror("malloc memory for array error!");**

**int\* buffer = data + data\_size;**

**/\***

**\* 每个处理器上生成随机数用于排序**

**\* 原程序中输出是乱序的，需要进行处理，方法和第一次作业类似**

**\*/**

**if (id != 0) {**

**MPI\_Recv(&mutex, 1, MPI\_CHAR, id - 1,**

**OUTPUT\_TYPE, MPI\_COMM\_WORLD, status);**

**}**

**/\***

**\* 用自身 id 作为随机数种子**

**\* 确保每次生成的数据相同，便与调试**

**\*/**

**srand(id);**

**/\***

**\* 需要注意的是，通过 MPI\_Recv 和 MPI\_Send 实现的串行输出只保证调用顺序**

**\* 而在 mpirun 中，多个处理器上的 io 顺序并不是调用顺序**

**\* 因此即使按顺序调用了 printf 输出顺序也有可能是错误的**

**\* 需要保证输出有序，可将数据发送到一个处理器，统一输出**

**\***

**\* 此外，经过实验可以发现 mpirun 会倾向于优先满足短输出**

**\* 且 mpirun 的 io 是非抢占的，即在一个 printf 执行的过程中，它不会被打断**

**\***

**\* 在保证调用顺序后，此处的输出往往是有序的**

**\* 这可能是因为各个处理器上的输出量差别不大，因此 mpirun 按照调用顺序输出**

**\*/**

**printf("----> This is node %d \n", id);**

**printf("On node %d the input data is:\n", id);**

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

**data[i] = (int)random() % 0xffff - 0x8000;**

**printf("%d ", data[i]);**

**}**

**printf("\n");**

**if (id != total - 1) {**

**MPI\_Send(&mutex, 1, MPI\_CHAR, id + 1,**

**OUTPUT\_TYPE, MPI\_COMM\_WORLD);**

**}**

**/\***

**\* 已清除部分没有必要的 MPI\_Barrier 以提高性能**

**\* 每个处理器将自己的 n / P 个数据串行快速排序**

**\* 对应于算法 13.5 步骤 ( 1 )**

**\*/**

**quick\_sort(data, 0, length - 1);**

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

**// ------------------------------------------------------------------**

**if (split) {**

**/\***

**\* 处理器不止一个，此时有并行正则采样的过程**

**\* 申请数组 sample 存放自身选取的 split 个元素**

**\* 申请数组 index 存放此处理器的开始位点和结束位点**

**\*/**

**int\* sample = (int\*)malloc(total \* split \* sizeof(int));**

**if (sample == 0) merror("malloc memory for sample error!");**

**int\* index = (int\*)malloc(total \* sizeof(int) \* 2);**

**if (index == 0) merror("malloc memory for index error!");**

**/\***

**\* 每个处理器从排好序的序列中选取第 w, 2w, 3w, …, (P - 1)w**

**\* 共 P - 1 个数据作为代表元素，其中 w = n / P \* P**

**\* 对应于算法 13.5 步骤 ( 2 )**

**\*/**

**for (int i = 0, n = length / total; i < split; i++)**

**sample[i] = data[(i + 1) \* n - 1];**

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

**if (id == 0) {**

**/\***

**\* 每个处理器将选好的代表元素送到处理器 P0 中**

**\* 对应于算法 13.5 步骤 ( 3 )**

**\*/**

**for (int i = 1, j = 0; i < total; i++) {**

**/\***

**\* MPI\_Irecv 是非阻塞接受，可在没接收完的时候继续接受新的**

**\* int MPI\_Irecv(**

**\* void \*buf,**

**\* int count,**

**\* MPI\_Datatype datatype,**

**\* int source,**

**\* int tag,**

**\* MPI\_Comm comm,**

**\* MPI\_Request \*request**

**\* )**

**\* 用 request 保存接受结构体，在 MPI\_Waitall 中判断是否完成接收**

**\*/**

**MPI\_Irecv(sample + i \* split, (int) (split \* sizeof(int)), MPI\_CHAR,**

**i, ALLTOONE\_TYPE + i, MPI\_COMM\_WORLD, request + j++);**

**}**

**/\* 等待全部数据完成传送 \*/**

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

**/\* 第一次同步：发送完成，0 号处理器接收完成 \*/**

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

**/\***

**\* 处理器 P0 将上一步送来的 P 段有序的数据序列做 P 路归并**

**\* 再选择排序后的第 P - 1, 2 \* (P - 1), …, (P - 1) \* (P - 1) 个共 P - 1 个主元**

**\* 对应于算法 13.5 步骤 ( 3 )**

**\***

**\* 这里改为采用串行快速排序**

**\* 一方面快速排序也可以使数据有序，供接下来的过程使用**

**\* 另一方面，排序数据的个数是 P \* P 的量级，相对 data\_size 不大**

**\* 因此使用快速排序也有较高的效率**

**\*/**

**quick\_sort(sample, 0, total \* split - 1);**

**/\* 第二次同步：0 号处理器排序完成 \*/**

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

**for (int i = 1; i < total; i++)**

**sample[i] = sample[i \* split - 1];**

**/\***

**\* 处理器 P0 将这 P - 1 个主元播送到所有处理器中**

**\* 对应于算法 13.5 步骤 ( 4 )**

**\*/**

**MPI\_Bcast(sample, (int)(total \* sizeof(int)), MPI\_CHAR, 0, MPI\_COMM\_WORLD);**

**/\* 第三次同步：0 号处理器广播，其它处理器接收完成 \*/**

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

**}**

**else {**

**/\* 将当前处理器选出的代表元素发送到 0 号处理器 \*/**

**MPI\_Send(sample, (int)(split \* sizeof(int)), MPI\_CHAR, 0, ALLTOONE\_TYPE + id, MPI\_COMM\_WORLD);**

**/\* 第一次同步：发送完成，0 号处理器接收完成 \*/**

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

**/\* 第二次同步：0 号处理器排序完成 \*/**

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

**MPI\_Bcast(sample, (int)(total \* sizeof(int)), MPI\_CHAR, 0, MPI\_COMM\_WORLD);**

**/\* 第三次同步：0 号处理器广播，其它处理器接收完成 \*/**

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

**}**

**/\***

**\* 每个处理器根据上步送来的 P - 1 个主元把自己的 n / P 个数据分成 P 段**

**\* 记为处理器 Pi 的第 j + 1 段，其中 i = 0, …, P - 1且 j = 0, …, P - 1**

**\* 对应于算法 13.5 步骤 ( 5 )**

**\*/**

**int current = 1;**

**/\* 对小于首个元素的主元，将它的开始位点和结束位点清零 \*/**

**for (index[0] = 0; data[0] >= sample[current] && current < total; current++)**

**index[2 \* current] = index[2 \* current - 1] = 0;**

**/\* 如果所有元素大于最大主元，直接将所有元素放在最后一段 \*/**

**if (current == total) index[2 \* current - 1] = length;**

**for (int l = 0; current < total; current++) {**

**/\***

**\* 原程序变量命名过于阴间**

**\* 已全部重命名，以便望文生义**

**\* 二分查找出比当前主元大的最小元素的位置存入 mid**

**\***

**\* 一个更好的处理是让 data[l] <= sample[current]，data[r] > sample[current]**

**\* 在此基础上二分缩小 l 和 r 直到 r = l + 1，此时 r 即为所求**

**\* 按此方法处理无需额外的循环对 mid 进行修正**

**\*/**

**int r = length, mid = (l + r) / 2;**

**while (data[mid] != sample[current] && l < r) {**

**if (data[mid] > sample[current]) r = mid - 1;**

**else l = mid + 1;**

**mid = (l + r) / 2;**

**}**

**while (data[mid] <= sample[current] && mid < length) mid++;**

**if (mid == length) {**

**/\***

**\* 当前段是最后一段，更新结束位点**

**\* 后面的各段将不再有意义，将它们的下标开始位点和结束位点清零**

**\* 最后更新 current 为 total 跳出循环**

**\*/**

**index[2 \* current - 1] = length;**

**for (int i = current; i < total; i++)**

**index[2 \* i] = index[2 \* i + 1] = 0;**

**current = total;**

**}**

**else {**

**/\***

**\* 上一段的结束位点是 mid**

**\* 下一段的开始位点是 mid**

**\*/**

**index[2 \* current] = mid;**

**index[2 \* current - 1] = mid;**

**}**

**/\* 更新二分开始的位置，进行新一轮二分 \*/**

**l = mid;**

**}**

**/\* 更新最后一段的结束位点 \*/**

**if (current == total) index[2 \* current - 1] = length;**

**/\* 各处理器完成分段 \*/**

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

**// ------------------------------------------------------------------**

**/\***

**\* 每个处理器发送它的第 i + 1 段给处理器 Pi**

**\* 从而使得第 i 个处理器含有所有处理器的第 i 段数据 (i = 0, …, P - 1)**

**\* 对应于算法 13.5 步骤 ( 6 )**

**\*/**

**for (int i = 0, j = 0; i < total; i++) {**

**if (i == id) {**

**/\***

**\* 是本处理器，需要发送数据**

**\* 此处复用了 sample 数组，存放各段的长度**

**\* 自己的 sample[i] 直接计算即可，不需要通过 MPI\_Recv 得到**

**\*/**

**sample[i] = index[2 \* i + 1] - index[2 \* i];**

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

**if (k != id) {**

**/\* 向第 k 个处理器发送本机第 k + 1 段的长度 \*/**

**int l = index[2 \* k + 1] - index[2 \* k];**

**MPI\_Send(&l, sizeof(int), MPI\_CHAR, k, MULTI\_LEN + id, MPI\_COMM\_WORLD);**

**}**

**}**

**else {**

**/\* 从第 i 个处理器获取其第 i + 1 段的长度，存入 sample[i] \*/**

**MPI\_Recv(sample + i, sizeof(int), MPI\_CHAR, i,**

**MULTI\_LEN + i, MPI\_COMM\_WORLD, status + j++);**

**}**

**}**

**/\* 发送完成，进行同步 \*/**

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

**// ------------------------------------------------------------------**

**/\***

**\* 每个处理器向其它处理器发送对应的数据**

**\* 第 i 个处理器向第 j 个处理器发送第 j 段**

**\* 用 length 维护当前处理器接受的数据的数量**

**\*/**

**length = 0;**

**for (int i = 0, j = 0; i < total; i++) {**

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

**if (i == id) {**

**/\* 当前处理器无需给自己发送，直接复制即可 \*/**

**for (int k = index[2 \* i]; k < index[2 \* i + 1]; k++)**

**buffer[length++] = data[k];**

**}**

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

**if (i == id) {**

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

**if (k != id) {**

**/\* 向其它处理器 k 发送对应的第 k 段 \*/**

**int count = index[2 \* k + 1] - index[2 \* k];**

**MPI\_Send(data + index[2 \* k], (int)(count \* sizeof(int)),**

**MPI\_CHAR, k, MULTI\_TYPE + id, MPI\_COMM\_WORLD);**

**}**

**}**

**else {**

**/\* 接收来自其它处理器的第 i 段，并维护 length \*/**

**MPI\_Recv(buffer + length, (int)(sample[i] \* sizeof(int)), MPI\_CHAR,**

**i , MULTI\_TYPE + i, MPI\_COMM\_WORLD, status + j++);**

**length = length + sample[i];**

**}**

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

**}**

**/\***

**\* 已清除部分没有必要的 MPI\_Barrier 以提高性能**

**\* 每个处理器再通过 P 路归并排序将上一步的到的数据排序**

**\* 从而这 n 个数据便是有序的**

**\* 对应于算法 13.5 步骤 ( 7 )**

**\***

**\* 此处的 data 是实质上的 buffer**

**\* 此处的 buffer 是实质上的 data**

**\* 归并的结果最终存在 buffer**

**\*/**

**multi\_merge(buffer, data, sample, total);**

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

**}**

**// ------------------------------------------------------------------**

**if (id != 0) {**

**MPI\_Recv(&mutex, 1, MPI\_CHAR, id - 1,**

**OUTPUT\_TYPE, MPI\_COMM\_WORLD, status);**

**}**

**/\***

**\* 此处同样存在输出顺序问题，通过 MPI\_Recv 和 MPI\_Send 保证 printf 的调用顺序并不能保证各处理器的输出顺序**

**\* 各个处理器上的输出量差别较大，会触发 mpirun 优先满足短输出的调度**

**\* 这显然不便于我们观察结果，例如可能出现 (#) ：**

**\* 某个处理器正在循环输出经过排序的数字**

**\* 另一个处理器输出 "On node %d the sorted data is : \n"**

**\* 此处理器继续循环输出经过排序的数字**

**\* 为此可以利用 mpirun 中 io 的非抢占性进行优化，具体的：**

**\* 申请一个缓冲区数组，将输出以字符串的形式存放在缓冲区**

**\* 一次性输出缓冲区，因为是单次输出，不会被打断**

**\* 虽然可能仍有输出不按顺序，但不再会出现上述 (#) 的情况，观察体验大大增加**

**\*/**

**char\* s = (char\*)malloc(16 \* data\_size);**

**sprintf(s, "# On node %d the sorted data is : \n", id);**

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

**if (i + 1 != length) sprintf(s, "%s %d ", s, buffer[i]);**

**else sprintf(s, "%s %d\n", s, buffer[i]);**

**}**

**fputs(s, stdout), free(s);**

**if (id != total - 1) {**

**MPI\_Send(&mutex, 1, MPI\_CHAR, id + 1,**

**OUTPUT\_TYPE, MPI\_COMM\_WORLD);**

**}**

**}**

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

**/\* 当前算法需要保证数据总数时处理器的个数的倍数 \*/**

**int id, data\_size = 16 \* 64;**

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

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

**if (id == 0) printf("Data Size : %d\n\n", data\_size);**

**psrs\_main(data\_size);**

**/\* 输出换行前同步一次，避免非 0 号处理器还没有输出完而换行已输出 \*/**

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

**if (id == 0) printf("\n");**

**MPI\_Finalize();**

**return 0;**

**}**

## 并行机上运行程序

**Data Size : 1024**

**----> This is node 0**

**On node 0 the input data is:**

**12530 -10687 31909 11942 -12030 -2488 16178 15169 -15689 17333 5589 -22486 -3933 -8014 -23610 -10612 -11813 16018 -25996 -2647 -23247 -30465 24439 10876 -23676 -10068 -30261 80 -15856 29582 -31119 -3326 -13871 790 8616 -25901 31070 -7972 -10732 -17386 9361 27625 25663 -27339 19611 2054 27585 -24969 18072 -31178 -27615 27593 3893 29592 5703 12985 19524 -24558 -19702 3668 -27743 -18053 -32424 -8846**

**----> This is node 1**

**On node 1 the input data is:**

**12530 -10687 31909 11942 -12030 -2488 16178 15169 -15689 17333 5589 -22486 -3933 -8014 -23610 -10612 -11813 16018 -25996 -2647 -23247 -30465 24439 10876 -23676 -10068 -30261 80 -15856 29582 -31119 -3326 -13871 790 8616 -25901 31070 -7972 -10732 -17386 9361 27625 25663 -27339 19611 2054 27585 -24969 18072 -31178 -27615 27593 3893 29592 5703 12985 19524 -24558 -19702 3668 -27743 -18053 -32424 -8846**

**----> This is node 2**

**On node 2 the input data is:**

**29107 24866 12705 12507 -662 -28715 5011 -12116 27227 -30056 5474 -16739 -13304 1455 -15861 -12524 14796 -16040 -20681 8727 3984 -28851 11361 22332 -841 -19837 31062 1977 -25562 -11690 -6980 -29222 -19591 -27041 16053 -20252 9779 21064 400 4239 -8991 5874 -12500 10473 -25438 4407 30718 22126 -11633 10037 -1914 -7648 -18814 9448 -18083 13113 -10389 12980 15090 -3183 -31477 -24656 -32405 14468**

**----> This is node 3**

**On node 3 the input data is:**

**5653 -8803 -30935 -19973 -8243 -15868 21994 -18082 31133 -24709 -2995 2936 22930 1841 -26836 27733 5551 -22633 -4792 -518 -18313 -10976 -5275 -27560 31925 -17376 -29328 12834 -26065 -2350 -31459 12356 -11152 3141 -7617 13373 20042 -18390 -4708 -14360 22436 25065 21344 12599 -5860 -5491 -25202 32459 4644 2774 31941 -13669 24566 -6101 -8460 23724 -23477 27747 -28977 -16774 25397 -27667 -4417 -18522**

**----> This is node 4**

**On node 4 the input data is:**

**29483 -7335 12731 -2712 11360 29234 -12203 -22060 -28155 -16098 -3039 -3809 21446 16596 -20200 -1859 30929 23483 12433 -30144 -20406 -24 5695 31921 11755 -18520 23449 -18061 30832 -30418 28357 -5220 -4984 -24447 24836 6376 -27979 -20134 17085 -23366 29303 14046 5593 17982 -2124 -14607 16124 28805 -23891 -4210 31429 -11529 28534 -28411 20393 -25246 -14162 -21693 -10539 -16097 -19342 17819 -21317 -24326**

**----> This is node 5**

**On node 5 the input data is:**

**-32698 11727 -26243 -1466 2643 22049 15897 -18944 21895 -26212 8803 24615 28507 29680 27497 -18003 13472 -31322 7010 12858 29841 -25288 -28529 21820 31528 -17253 -4900 18845 -29721 10574 10108 -29651 22301 -16135 1651 -7822 -26853 17548 6002 -4958 24105 -17961 -13110 -12923 -21048 -18380 -30926 -7576 15833 8852 -27485 12907 -16436 -23245 1960 -17675 25037 -2940 -31596 -4684 7634 11280 -1567 -2831**

**----> This is node 6**

**On node 6 the input data is:**

**-24557 -17683 23669 -32313 1425 -3948 20618 27565 1844 -13015 -16293 31610 -10928 -12780 4184 -25512 -2933 9334 -5750 26021 -23222 -6153 -3403 15019 19461 -30942 -3758 -19855 4644 -20273 30932 -19913 27579 -10934 -19457 -3762 -14882 1161 23803 19731 -11853 -25256 -14194 -22781 27499 22758 -15524 -8200 -674 11494 17821 -23896 5341 14419 23891 -7965 16245 20133 -27820 20890 32628 -29655 -31790 -5327**

**----> This is node 7**

**On node 7 the input data is:**

**-25016 -11939 -11499 14378 -25668 21760 15093 -8281 -7088 4430 18912 10073 -3652 -3512 -13453 14002 16775 -20907 -3738 25063 20181 -20520 21535 10026 -25894 13376 19598 -21851 -23206 11442 -17270 -15454 32272 3999 31693 -26163 25759 -18749 -1675 -14095 18449 17237 28746 14797 13725 15294 -3968 -2266 -5613 -7706 -9970 14568 4543 -21202 -8173 11417 -7826 -21342 -10433 1737 -9900 5065 19051 -10395**

**----> This is node 8**

**On node 8 the input data is:**

**-3937 10921 3604 -24549 21628 -1900 -2056 -19152 1984 28195 25439 -28952 3248 -5601 -23828 -29874 -6113 22385 -8828 28734 -11588 -9295 28024 1274 -13108 -30956 -2334 -5121 12977 22135 70 -23726 -32478 3674 -15507 -10850 1775 -17563 -30001 -29008 10632 -4562 7575 -18886 22605 -16253 16775 16493 -26634 -24820 -20308 27313 31421 -25051 28587 -14454 -23238 -6514 -19574 -10261 15621 -19504 -1219 -16857**

**----> This is node 9**

**On node 9 the input data is:**

**29312 -28164 -24913 27974 17322 6120 1647 -11920 -8691 -16476 29603 -23890 -25090 -21199 7109 28302 15666 -1493 13958 16601 16237 -24153 28613 26917 5551 24396 -17564 8415 -4099 23936 9302 -7554 28540 -15611 20420 -19672 23277 22067 -31592 14586 -27176 30780 -9304 -19498 9581 -2194 -23963 -7520 29081 -10004 9082 12551 31378 4928 6701 -28606 -3442 -10862 12577 -7541 -19693 21879 -15095 8847**

**----> This is node 10**

**On node 10 the input data is:**

**17627 -15925 -15645 16947 5515 -26811 -19623 -5152 -22071 -9666 28110 -7830 31015 190 -30684 -16552 -9938 18695 22458 11710 -9470 -10024 -9013 -2002 24059 -24450 17024 23330 -22000 5653 22459 28395 -10271 6815 -20193 28012 12772 -7048 22861 23469 16055 18204 -17128 -18465 18394 17723 30518 -24310 3652 20209 -12600 26950 10185 -21613 24948 -31291 -13295 9205 24807 -2526 -17908 -18268 25869 4589**

**----> This is node 11**

**On node 11 the input data is:**

**29265 6906 -29156 12117 21670 -15778 4688 -6817 -6234 29420 1557 -23072 -28830 -27759 -25947 -4624 443 -7584 22880 -24470 23125 -23269 11384 -23212 -24063 30987 -5551 29115 -22 23165 -4092 29243 30071 -480 8594 -13793 16510 13282 12158 10277 9935 -19051 19973 -18895 -14042 -5974 9250 -13598 19210 32130 27467 9569 -23906 6084 19125 17566 -28464 13574 13914 4282 -28796 -22945 759 -31491**

**----> This is node 12**

**On node 12 the input data is:**

**29022 30441 -14994 -13270 9560 -2408 -5799 2230 9134 -1454 31685 -25919 -31231 -1449 15665 -1564 -651 -12946 -29526 7082 -15781 3015 -12098 -11179 7645 10409 -28119 813 -27983 6444 22135 -31728 -28650 -25625 20537 13679 -28033 -18029 15909 13869 -19482 14827 -12049 -17945 -19389 -29151 -19509 -20039 -9329 -16266 -12957 -25110 -13251 7713 29247 27162 -14645 1128 27976 -9860 -25195 -15424 23948 11691**

**----> This is node 13**

**On node 13 the input data is:**

**17572 11568 -11399 -30412 -24229 -13879 -14046 -13368 16536 -5952 -532 -20359 -4981 28471 -31617 24845 6769 4190 1935 -13784 -11736 -29425 -24844 23568 -2750 31181 18907 7483 -15360 26756 24973 2212 5557 13575 4569 -18671 -304 23291 -32039 16232 17339 198 28642 -20409 -4098 29793 4436 2671 -31551 -26395 21655 22248 9715 29580 13049 -25801 27994 31956 -18318 -20132 25945 6655 -17920 31503**

**----> This is node 14**

**On node 14 the input data is:**

**-28405 -4296 -32072 18754 -11320 300 22599 -31673 15976 25568 17710 -5599 25822 -32558 -32487 -28563 -10467 7653 -32458 4323 -23283 21371 10906 -11420 -16438 20849 -22063 -19324 -15570 -21755 -31076 21560 -26050 -30380 -25221 -4602 -30080 30146 -3507 -14103 22948 -18564 13066 -16765 -18353 -19421 -12560 -28820 -11768 -12250 8271 30484 9122 -13590 -13703 25452 7259 29769 -26639 -8310 -24752 7820 13250 14733**

**----> This is node 15**

**On node 15 the input data is:**

**28740 -27248 -716 -21768 -20697 -15278 -21229 27034 25910 7070 10985 29624 -23679 -18054 20462 -13613 -6834 -3046 -18459 1160 -22438 19887 24355 -29552 -30193 5228 18935 -16082 24617 11710 22812 -12178 -15538 22096 -1178 29300 -25948 10362 23568 32730 17432 1786 -3180 -6247 -16268 -15485 12908 -23101 -18531 -5550 -21941 -8200 14337 -30353 27783 16912 -25124 13951 -31937 32261 25662 -9125 -12683 -22643**

**# On node 0 the sorted data is :**

**-32698 -32558 -32487 -32478 -32458 -32424 -32424 -32405 -32313 -32072 -32039 -31937 -31790 -31728 -31673 -31617 -31596 -31592 -31551 -31491 -31477 -31459 -31322 -31291 -31231 -31178 -31178 -31119 -31119 -31076 -30956 -30942 -30935 -30926 -30684 -30465 -30465 -30418 -30412 -30380 -30353 -30261 -30261 -30193 -30144 -30080 -30056 -30001 -29874 -29721 -29655 -29651 -29552 -29526 -29425 -29328 -29222 -29156 -29151 -29008 -28977 -28952 -28851 -28830 -28820 -28796 -28715 -28650 -28606 -28563 -28529 -28464 -28411 -28405 -28164 -28155 -28119 -28033 -27983**

**# On node 1 the sorted data is :**

**-27979 -27820 -27759 -27743 -27743 -27667 -27615 -27615 -27560 -27485 -27339 -27339 -27248 -27176 -27041 -26853 -26836 -26811 -26639 -26634 -26395 -26243 -26212 -26163 -26065 -26050 -25996 -25996 -25948 -25947 -25919 -25901 -25901 -25894 -25801 -25668 -25625 -25562 -25512 -25438 -25288 -25256 -25246 -25221 -25202 -25195 -25124 -25110 -25090 -25051 -25016 -24969 -24969 -24913 -24844 -24820 -24752 -24709 -24656 -24558 -24558 -24557**

**# On node 2 the sorted data is :**

**-24549 -24470 -24450 -24447 -24326 -24310 -24229 -24153 -24063 -23963 -23906 -23896 -23891 -23890 -23828 -23726 -23679 -23676 -23676 -23610 -23610 -23477 -23366 -23283 -23269 -23247 -23247 -23245 -23238 -23222 -23212 -23206 -23101 -23072 -22945 -22781 -22643 -22633 -22486 -22486 -22438 -22071 -22063 -22060 -22000 -21941 -21851 -21768 -21755 -21693 -21613 -21342 -21317 -21229 -21202**

**# On node 3 the sorted data is :**

**-21199 -21048 -20907 -20697 -20681 -20520 -20409 -20406 -20359 -20308 -20273 -20252 -20200 -20193 -20134 -20132 -20039 -19973 -19913 -19855 -19837 -19702 -19702 -19693 -19672 -19623 -19591 -19574 -19509 -19504 -19498 -19482 -19457 -19421 -19389 -19342 -19324 -19152 -19051 -18944 -18895 -18886 -18814 -18749 -18671 -18564 -18531 -18522 -18520 -18465 -18459 -18390 -18380 -18353 -18318 -18313 -18268 -18083 -18082 -18061 -18054 -18053 -18053 -18029 -18003 -17961 -17945 -17920 -17908 -17683**

**# On node 4 the sorted data is :**

**-17675 -17564 -17563 -17386 -17386 -17376 -17270 -17253 -17128 -16857 -16774 -16765 -16739 -16552 -16476 -16438 -16436 -16293 -16268 -16266 -16253 -16135 -16098 -16097 -16082 -16040 -15925 -15868 -15861 -15856 -15856 -15781 -15778 -15689 -15689 -15645 -15611 -15570 -15538 -15524 -15507 -15485 -15454 -15424 -15360 -15278 -15095 -14994 -14882 -14645 -14607 -14454 -14360 -14194 -14162 -14103 -14095 -14046 -14042 -13879 -13871 -13871 -13793 -13784 -13703 -13669 -13613**

**# On node 5 the sorted data is :**

**-13598 -13590 -13453 -13368 -13304 -13295 -13270 -13251 -13110 -13108 -13015 -12957 -12946 -12923 -12780 -12683 -12600 -12560 -12524 -12500 -12250 -12203 -12178 -12116 -12098 -12049 -12030 -12030 -11939 -11920 -11853 -11813 -11813 -11768 -11736 -11690 -11633 -11588 -11529 -11499 -11420 -11399 -11320 -11179 -11152 -10976**

**# On node 6 the sorted data is :**

**-10934 -10928 -10862 -10850 -10732 -10732 -10687 -10687 -10612 -10612 -10539 -10467 -10433 -10395 -10389 -10271 -10261 -10068 -10068 -10024 -10004 -9970 -9938 -9900 -9860 -9666 -9470 -9329 -9304 -9295 -9125 -9013 -8991 -8846 -8846 -8828 -8803 -8691 -8460 -8310 -8281 -8243 -8200 -8200 -8173 -8014 -8014 -7972 -7972 -7965 -7830 -7826 -7822 -7706 -7648 -7617 -7584 -7576 -7554 -7541 -7520 -7335 -7088 -7048 -6980 -6834 -6817 -6514 -6247 -6234 -6153**

**# On node 7 the sorted data is :**

**-6113 -6101 -5974 -5952 -5860 -5799 -5750 -5613 -5601 -5599 -5551 -5550 -5491 -5327 -5275 -5220 -5152 -5121 -4984 -4981 -4958 -4900 -4792 -4708 -4684 -4624 -4602 -4562 -4417 -4296 -4210 -4099 -4098 -4092 -3968 -3948 -3937 -3933 -3933 -3809 -3762 -3758 -3738 -3652 -3512 -3507 -3442 -3403 -3326 -3326 -3183 -3180 -3046 -3039 -2995**

**# On node 8 the sorted data is :**

**-2940 -2933 -2831 -2750 -2712 -2647 -2647 -2526 -2488 -2488 -2408 -2350 -2334 -2266 -2194 -2124 -2056 -2002 -1914 -1900 -1859 -1675 -1567 -1564 -1493 -1466 -1454 -1449 -1219 -1178 -841 -716 -674 -662 -651 -532 -518 -480 -304 -24 -22 70 80 80 190 198 300 400 443 759 790 790 813 1128**

**# On node 9 the sorted data is :**

**1160 1161 1274 1425 1455 1557 1647 1651 1737 1775 1786 1841 1844 1935 1960 1977 1984 2054 2054 2212 2230 2643 2671 2774 2936 3015 3141 3248 3604 3652 3668 3668 3674 3893 3893 3984 3999 4184 4190 4239 4282 4323 4407 4430 4436 4543 4569 4589 4644 4644 4688 4928 5011 5065 5228 5341 5474 5515 5551 5551**

**# On node 10 the sorted data is :**

**5557 5589 5589 5593 5653 5653 5695 5703 5703 5874 6002 6084 6120 6376 6444 6655 6701 6769 6815 6906 7010 7070 7082 7109 7259 7483 7575 7634 7645 7653 7713 7820 8271 8415 8594 8616 8616 8727 8803 8847 8852 9082 9122 9134 9205 9250 9302 9334 9361 9361 9448 9560 9569 9581 9715 9779 9935 10026 10037 10073**

**# On node 11 the sorted data is :**

**10108 10185 10277 10362 10409 10473 10574 10632 10876 10876 10906 10921 10985 11280 11360 11361 11384 11417 11442 11494 11568 11691 11710 11710 11727 11755 11942 11942 12117 12158 12356 12433 12507 12530 12530 12551 12577 12599 12705 12731 12772 12834 12858 12907 12908 12977 12980 12985 12985 13049 13066 13113 13250 13282 13373 13376 13472 13574 13575 13679 13725**

**# On node 12 the sorted data is :**

**13869 13914 13951 13958 14002 14046 14337 14378 14419 14468 14568 14586 14733 14796 14797 14827 15019 15090 15093 15169 15169 15294 15621 15665 15666 15833 15897 15909 15976 16018 16018 16053 16055 16124 16178 16178 16232 16237 16245 16493 16510 16536 16596 16601 16775 16775 16912 16947 17024 17085 17237 17322 17333 17333 17339 17432 17548 17566 17572 17627 17710**

**# On node 13 the sorted data is :**

**17723 17819 17821 17982 18072 18072 18204 18394 18449 18695 18754 18845 18907 18912 18935 19051 19125 19210 19461 19524 19524 19598 19611 19611 19731 19887 19973 20042 20133 20181 20209 20393 20420 20462 20537 20618 20849 20890 21064 21344 21371 21446 21535 21560 21628 21655 21670 21760 21820 21879 21895 21994 22049 22067 22096 22126 22135 22135 22248 22301 22332 22385 22436 22458 22459 22599 22605 22758 22812**

**# On node 14 the sorted data is :**

**22861 22880 22930 22948 23125 23165 23277 23291 23330 23449 23469 23483 23568 23568 23669 23724 23803 23891 23936 23948 24059 24105 24355 24396 24439 24439 24566 24615 24617 24807 24836 24845 24866 24948 24973 25037 25063 25065 25397 25439 25452 25568 25662 25663 25663 25759 25822 25869 25910 25945 26021 26756**

**# On node 15 the sorted data is :**

**26917 26950 27034 27162 27227 27313 27467 27497 27499 27565 27579 27585 27585 27593 27593 27625 27625 27733 27747 27783 27974 27976 27994 28012 28024 28110 28195 28302 28357 28395 28471 28507 28534 28540 28587 28613 28642 28734 28740 28746 28805 29022 29081 29107 29115 29234 29243 29247 29265 29300 29303 29312 29420 29483 29580 29582 29582 29592 29592 29603 29624 29680 29769 29793 29841 30071 30146 30441 30484 30518 30718 30780 30832 30929 30932 30987 31015 31062 31070 31070 31133 31181 31378 31421 31429 31503 31528 31610 31685 31693 31909 31909 31921 31925 31941 31956 32130 32261 32272 32459 32628 32730**

## 实验结果

提交的为。