# 并行与分布式作业

“ 黄炜钊”

第五次作业

姓名： 黄炜钊

班级： 行政三班

学号： 18340066

## 一、 问题描述

1、 Consider a sparse matrix stored in the compressed row format (you may find a description of this format on the web or any suitable text on sparse linear algebra). Write an OpenMP program for computing the product of this matrix with a vector. Download sample matrices from the Matrix Market (http://math.nist.gov/MatrixMarket/) and test the performance of your implementation as a function of matrix size and number of threads.

2、 Implement a producer-consumer framework in OpenMP using sections to create a single producer task and a single consumer task. Ensure appropriate synchronization using locks. Test your program for a varying number of producers and consumers.

3、 利用 MPI 通信程序测试本地进程以及远程进程之间的通信时延和带宽。

## 二、 解决方案

1、

从网站上下载了 1138×1138，有 2596 个非零元素的稀疏矩

阵，向量是随机生成 1138 维的稠密向量，输出 100000 次乘法的时间进行展示。具体代码如下：

```
#include <bits/stdc++.h>
#include <omp.h>
using namespace std;
typedef double lf;
typedef vector<lf> Vec;
typedef vector<vector<pair<int, lf> > > Mat;
int m, n, th;
Vec operator*(const Vec &v, const Mat &m)
{
    Vec r(m.size(), 0);
#pragma omp parallel for num_threads(th)
    for (int i = 0; i < m.size(); ++i)
        for (const auto &p : m[i])
            r[i] += v[p.first] * p.second;
    return r;
}
int main()
{
    ifstream fin("1138_bus.mtx");
    while (fin.peek() == '%')
        while (fin.get() != '\n')
            ;
    fin >> m >> n >> th;
    Mat ma(m);
    for (int x, y, i = 0; i < th; ++i)
    {
        lf t;
        fin >> x >> y >> t;
        ma[x - 1].emplace_back(y - 1, t);
    }
    Vec ve(n);
    for (int i = 0; i < n; ++i)
        ve[i] = rand();
//  while(1)
//{
        cout << "number of threads: ";
    cin >> th;
    auto begin = std::chrono::system_clock::now();
    for (int i = 1e5; i; --i)
        ve *ma;
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - begin;
    std::cout << "elapsed time: " << elapsed_seconds.count() << "
s\n";

//}
system("pause");
return 0;
}
```

编译运行得到三组数据。

2、代码如下：

main.cpp:

```
#include <chrono>
#include <iostream>
#include "my.hpp"
Condor::MultiAccessQueue<int> q;
void producer(int cnt)
{
    for (int i = 0; i < cnt; ++i)
        q.push(i);
}
void consumer(int cnt)
{
    for (int i = 0; i < cnt; ++i)
        q.pop();
}
int main()
{
    int num;
    std::cout << "number of producer-consumers: ";
    std::cin >> num;
    auto begin = std::chrono::system_clock::now();
#pragma omp parallel for
    for (int i = 0; i < num; ++i)
        producer(1000);
#pragma omp parallel for
    for (int i = 0; i < num; ++i)
        consumer(1000);
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - begin;
    std::cout << "elapsed time: " << elapsed_seconds.count() << "s\n";
}
```

my.hpp:

```
#ifndef _my_hpp_
#define _my_hpp_
#include <queue>
#include <mutex>
namespace Condor
{
template <class T>
class MultiAccessQueue : std::queue<T>
{
private:
    mutable std::mutex mu;

public:
    void push(T val)
    {
        std::lock_guard<std::mutex> guard(mu);
        std::queue<T>::push(val);
    }
    void pop()
    {
        std::lock_guard<std::mutex> guard(mu);
        std::queue<T>::pop();
    }
    T front() const
    {
        std::lock_guard<std::mutex> guard(mu);
        return std::queue<T>::front();
    }
};
} // namespace Condor
#endif
```

编译运行，输入不同的数据得到不同的运行时间。

3、

基本思路是统计进程之间传输一定大小的数据所需要的时间来得到通信时延和带宽。

通信开始之前，将要用的节点的主机名称以及每个进程对

应的通信进程的 rank 发给 rank 为 0 的进程处。然后将所有的进程分成两半，小于总进程数一半的那部分的功能是统计数据发送、接收一个来回的时间并发送到 rank=0 的进程处，大于的那部分只需要接收并发送回去即可。同时 rank=0 的进程还对所有的时间信息进行统计、求平均。代码如下：

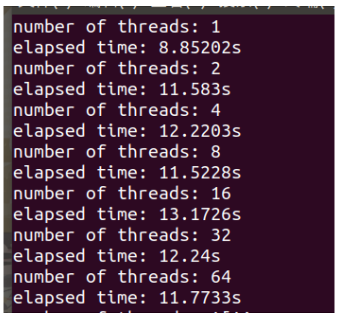
```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define BUF_SIZE 100000
#define MAX_TASKS 32
#define TEST_TIMES 32

int main(int argc, char* argv[]) {
    char buffer[BUF_SIZE] = {};
    memset(buffer, 'x', BUF_SIZE);
    char host_name[MPI_MAX_PROCESSOR_NAME];
    int host_name_length;
    char host_map[MAX_TASKS][MPI_MAX_PROCESSOR_NAME];
    memset(host_map, 0, MAX_TASKS * MPI_MAX_PROCESSOR_NAME);
    int partner_rank;
    int task_pair[MAX_TASKS] = {};
    memset(buffer, 0, MAX_TASKS);
    double timings[MAX_TASKS][4];
    memset(timings, 0, MAX_TASKS * 3);
    int tag = 1;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    if (world_size % 2 != 0) {
        printf("There must be an even number of tasks.\n");
        int error_code = 1;
        MPI_Abort(MPI_COMM_WORLD, error_code);
        exit(1);
    }
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Get_processor_name(host_name, &host_name_length);
    MPI_Gather(&host_name, MPI_MAX_PROCESSOR_NAME, MPI_CHAR, &host_map,
 MPI_MAX_PROCESSOR_NAME, MPI_CHAR, 0, MPI_COMM_WORLD);
    if (world_rank < world_size / 2) {
        partner_rank = world_size / 2 + world_rank;
    } else {
        partner_rank = world_rank - world_size / 2;
    }
    MPI_Gather(&partner_rank, 1, MPI_INT, &task_pair, 1, MPI_INT, 0, MP
I_COMM_WORLD);
    if (world_rank == 0) {
        puts("**********MPI BANDWIDTH TEST**********");
        printf("MPI world size: %d\n", world_size);
        printf("Test message size: %d\n", BUF_SIZE);
        printf("MPI_Wtick resolution: %e\n", MPI_Wtick());
        for (int i = 0; i < world_size; ++i) {
            printf("Task %d is on %s with partner %d\n", i, host_map[i]
, task_pair[i]);
        }
        puts("****************************************");
    }
    if (world_rank < world_size / 2) {
        double best_bw = .0, worst_bw = .99E+99, total_bw = .0;
        double total_time = .0;
        for (int i = 0; i < TEST_TIMES; ++i) {
            double nbytes = sizeof(char) * BUF_SIZE;
            double start_time = MPI_Wtime();
            MPI_Send(&buffer, BUF_SIZE, MPI_CHAR, partner_rank, tag, MP
I_COMM_WORLD);
            MPI_Recv(&buffer, BUF_SIZE, MPI_CHAR, partner_rank, tag, MP
I_COMM_WORLD, &status);
            double end_time = MPI_Wtime();
            double run_time = end_time - start_time;
            double bw = (2 * nbytes) / run_time;
            total_bw += bw;
            best_bw = bw > best_bw ? bw : best_bw;
            worst_bw = bw < worst_bw ? bw : worst_bw;
            total_time += run_time;
        }
        best_bw /= 1000000.0;
        worst_bw /= 1000000.0;
        double avg_bw = (total_bw / 1000000.0) / TEST_TIMES;
        total_time /= TEST_TIMES;
```

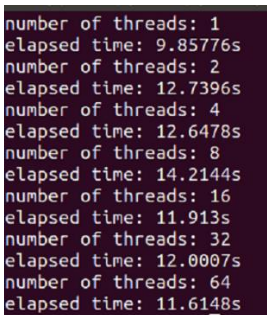
```c
        if (world_rank == 0) {
            timings[0][0] = best_bw;
            timings[0][1] = avg_bw;
            timings[0][2] = worst_bw;
            timings[0][3] = total_time;
            double best_all = .0, worst_all = .0, avg_all = .0;
            double time_all = .0;
            for (int j = 1; j < world_size / 2; ++j) {
                MPI_Recv(&timings[j], 4, MPI_DOUBLE, j, tag, MPI_COMM_WORLD, &status);
            }
            for (int j = 0; j < world_size / 2; ++j) {
                printf("Task pair: %d - %d: best: %lf, avg: %lf, worst: %lf, time: %lf\n", j, task_pair[j], timings[j][0], timings[j][1], timings[j][2], timings[j][3]);
                best_all += timings[j][0];
                avg_all += timings[j][1];
                worst_all += timings[j][2];
                time_all += timings[j][3];
            }
            printf("Total avg: best: %lf, avg: %lf, worst: %lf, time: %lf\n", best_all / (world_size / 2), avg_all / (world_size / 2), worst_all / (world_size / 2), time_all / (world_size / 2));
        } else {
            double tmp_timings[4];
            tmp_timings[0] = best_bw;
            tmp_timings[1] = avg_bw;
            tmp_timings[2] = worst_bw;
            tmp_timings[3] = total_time;
            MPI_Send(tmp_timings, 4, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD);
        }
    } else {
         for (int i = 0; i < TEST_TIMES; ++i) {
            MPI_Recv(&buffer, BUF_SIZE, MPI_CHAR, partner_rank, tag, MPI_COMM_WORLD, &status);
            MPI_Send(&buffer, BUF_SIZE, MPI_CHAR, partner_rank, tag, MPI_COMM_WORLD);
        }
    }
    MPI_Finalize();
    return 0;
}
```

## 三、 实验结果

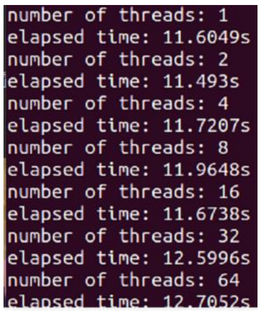
1、三组数据如下，取其平均值得到下表。



```
number of threads: 1
elapsed time: 8.85202s
number of threads: 2
elapsed time: 11.583s
number of threads: 4
elapsed time: 12.2203s
number of threads: 8
elapsed time: 11.5228s
number of threads: 16
elapsed time: 13.1726s
number of threads: 32
elapsed time: 12.24s
number of threads: 64
elapsed time: 11.7733s
```



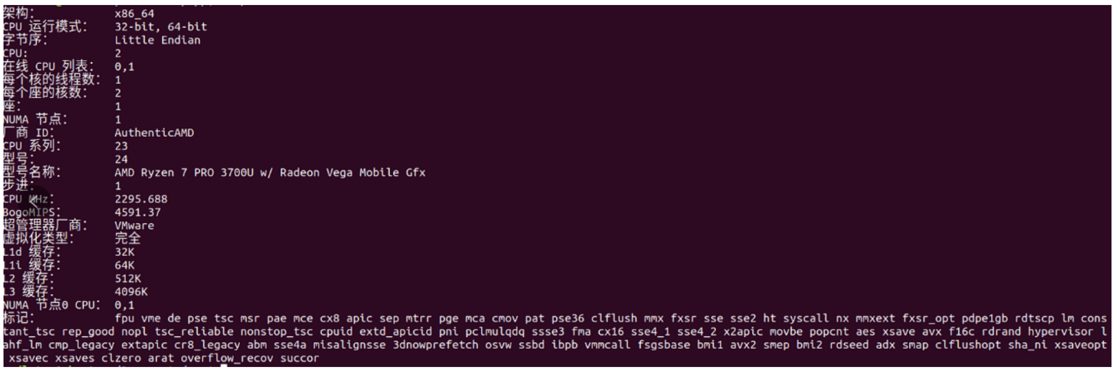
```
number of threads: 1
elapsed time: 9.85776s
number of threads: 2
elapsed time: 12.7396s
number of threads: 4
elapsed time: 12.6478s
number of threads: 8
elapsed time: 14.2144s
number of threads: 16
elapsed time: 11.913s
number of threads: 32
elapsed time: 12.0007s
number of threads: 64
elapsed time: 11.6148s
```



```
number of threads: 1
elapsed time: 11.6049s
number of threads: 2
elapsed time: 11.493s
number of threads: 4
elapsed time: 11.7207s
number of threads: 8
elapsed time: 11.9648s
number of threads: 16
elapsed time: 11.6738s
number of threads: 32
elapsed time: 12.5996s
number of threads: 64
elapsed time: 12.7052s
```

| number of threads | elapsed time |
|---|---|
| 1 | 10.10489 |
| 2 | 11.93853 |
| 4 | 12.19626 |
| 8 | 12.56733 |
| 16 | 12.25313 |
| 32 | 12.28010 |
| 64 | 12.05110 |

Cpu 的型号如下：

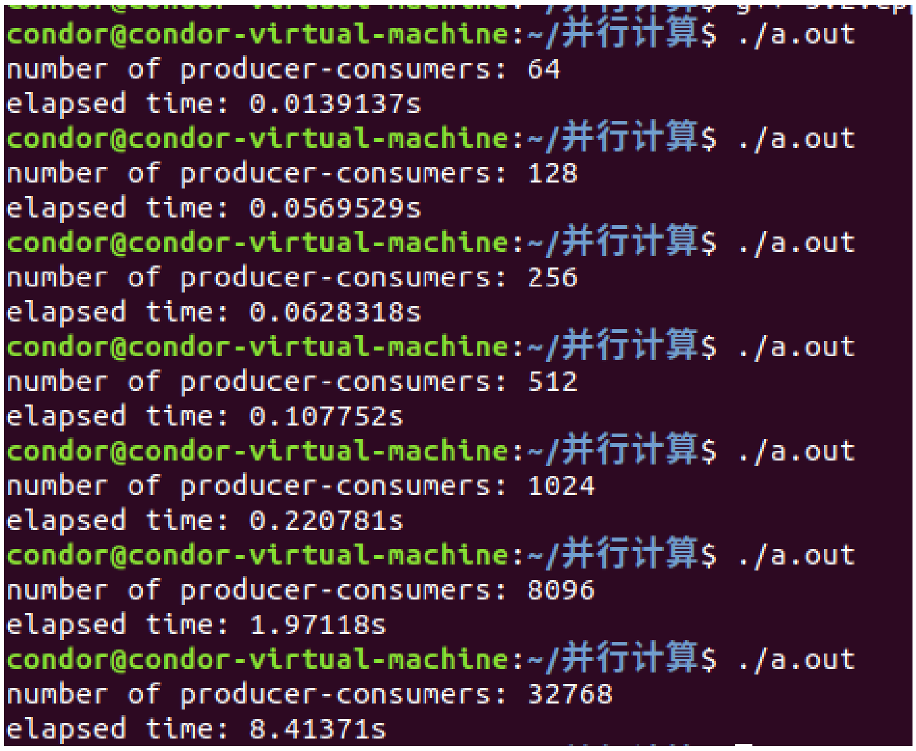


```
架构:            x86_64
CPU 运行模式:    32-bit, 64-bit
字节序:          Little Endian
CPU:             2
在线 CPU 列表:   0,1
每个核的线程数:  1
每个座的核数:    2
座:              1
NUMA 节点:       1
厂商 ID:         AuthenticAMD
CPU 系列:        23
型号:            24
型号名称:        AMD Ryzen 7 PRO 3700U w/ Radeon Vega Mobile Gfx
步进:            1
CPU MHz:         2295.688
BogoMIPS:        4591.37
超管理器厂商:    VMware
虚拟化类型:      完全
L1d 缓存:        32K
L1i 缓存:        64K
L2 缓存:         512K
L3 缓存:         4096K
NUMA 节点0 CPU:  0,1
标记:            fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm cons
tant_tsc rep_good nopl tsc_reliable nonstop_tsc cpuid extd_apicid pni pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor l
ahf_lm cmp_legacy extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw ssbd ibpb vmmcall fsgsbase bmi1 avx2 smep bmi2 rdseed adx smap clflushopt sha_ni xsaveopt
 xsavec xsaves clzero arat overflow_recov succor
```

由表格可以看出，线程数为 1 时具有最高的加速比，在线程数为 8 时加速比最低，随后线程数的增加减少的运行时间比把任务分配到多个线程消耗的资源要多，所以运行时间又降

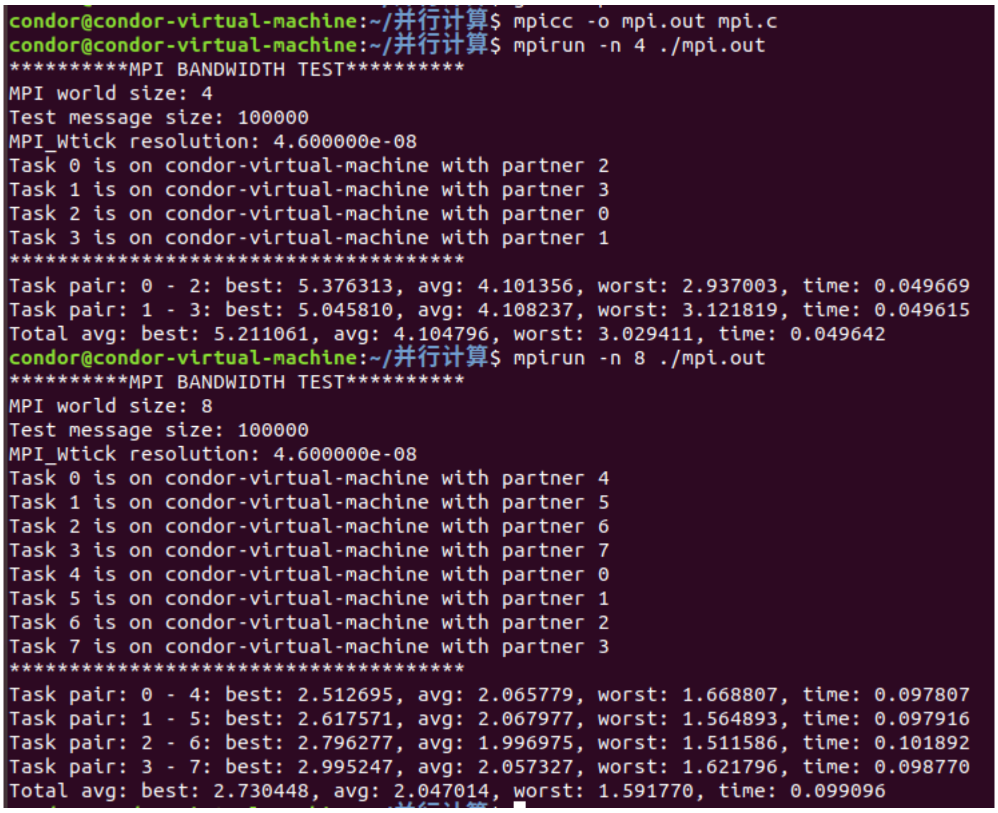
下来了。

2、得到不同的运行时间如下：



```
condor@condor-virtual-machine:~/并行计算$ ./a.out
number of producer-consumers: 64
elapsed time: 0.0139137s
condor@condor-virtual-machine:~/并行计算$ ./a.out
number of producer-consumers: 128
elapsed time: 0.0569529s
condor@condor-virtual-machine:~/并行计算$ ./a.out
number of producer-consumers: 256
elapsed time: 0.0628318s
condor@condor-virtual-machine:~/并行计算$ ./a.out
number of producer-consumers: 512
elapsed time: 0.107752s
condor@condor-virtual-machine:~/并行计算$ ./a.out
number of producer-consumers: 1024
elapsed time: 0.220781s
condor@condor-virtual-machine:~/并行计算$ ./a.out
number of producer-consumers: 8096
elapsed time: 1.97118s
condor@condor-virtual-machine:~/并行计算$ ./a.out
number of producer-consumers: 32768
elapsed time: 8.41371s
```

3、编译运行，得到结果如下：



```
condor@condor-virtual-machine:~/并行计算$ mpicc -o mpi.out mpi.c
condor@condor-virtual-machine:~/并行计算$ mpirun -n 4 ./mpi.out
**********MPI BANDWIDTH TEST**********
MPI world size: 4
Test message size: 100000
MPI_Wtick resolution: 4.600000e-08
Task 0 is on condor-virtual-machine with partner 2
Task 1 is on condor-virtual-machine with partner 3
Task 2 is on condor-virtual-machine with partner 0
Task 3 is on condor-virtual-machine with partner 1
**************************************
Task pair: 0 - 2: best: 5.376313, avg: 4.101356, worst: 2.937003, time: 0.049669
Task pair: 1 - 3: best: 5.045810, avg: 4.108237, worst: 3.121819, time: 0.049615
Total avg: best: 5.211061, avg: 4.104796, worst: 3.029411, time: 0.049642
condor@condor-virtual-machine:~/并行计算$ mpirun -n 8 ./mpi.out
**********MPI BANDWIDTH TEST**********
MPI world size: 8
Test message size: 100000
MPI_Wtick resolution: 4.600000e-08
Task 0 is on condor-virtual-machine with partner 4
Task 1 is on condor-virtual-machine with partner 5
Task 2 is on condor-virtual-machine with partner 6
Task 3 is on condor-virtual-machine with partner 7
Task 4 is on condor-virtual-machine with partner 0
Task 5 is on condor-virtual-machine with partner 1
Task 6 is on condor-virtual-machine with partner 2
Task 7 is on condor-virtual-machine with partner 3
**************************************
Task pair: 0 - 4: best: 2.512695, avg: 2.065779, worst: 1.668807, time: 0.097807
Task pair: 1 - 5: best: 2.617571, avg: 2.067977, worst: 1.564893, time: 0.097916
Task pair: 2 - 6: best: 2.796277, avg: 1.996975, worst: 1.511586, time: 0.101892
Task pair: 3 - 7: best: 2.995247, avg: 2.057327, worst: 1.621796, time: 0.098770
Total avg: best: 2.730448, avg: 2.047014, worst: 1.591770, time: 0.099096
```

由图可知，单节点通信传输 $10^5$ 个字节的数据，4 个进程传输时延平均为 0.049642s，带宽为 4.104796MBps; 8 个进程传输时延平均为 0.99096s，带宽为 2.047014MPBps。