

# 期末大作业实验

17341046 郭梓煜

## 实验目的

1. 根据 List Ranking 算法完成串行版本代码，分析串行版本代码的可并行性，将串行代码改写成 MPI 并行版本和 CUDA 并行版本，并比较三种版本代码的运行速度。  
输入：一个N元素的单向链表，该链表放在一个长度为N的向量中。链表的每一个元素（除链尾的一个）都有一个后继的数组下标。链尾的元素的后继为 '-1'。  
定义链表元素的序号（rank）为“其后继的总数”。例如链头的序号为N-1，链尾的序号为0。向量中的元素不一定按照链表的顺序存放。问题：要求算出每一个向量元素的序号。
2. 稀疏矩阵乘法，输入输出矩阵都用压缩行格式存储: 实现MPI和CUDA算法，并与串行程序比较；

## 实验环境

- 图形化管理文件: winscp
- 编辑器: vscode
- 语言: cuda, mpi
- 打开命令行界面: putty
- 编译器: nvcc
- 集群: 222.200.180.115
- source : /public/software/profile.d/cuda10.0.sh  
/public/software/profile.d/mpi\_openmpi-intel-2.1.2.sh

## 实验代码实现与分析

### 链表求序mpi实现

- MPI\_Init : MPI初始化 , MPI\_Finalize : 结束MPI程序的运行
- MPI\_Comm\_rank : 获取进程的进程号 , MPI\_Comm\_size : 获取进程个数
- MPI\_Send : 发送消息 , MPI\_Recv : 接收消息
- MPI\_Reduce : 归约操作, MPI\_Bcast: 广播, MPI\_barrier: 同步
- 按照老师给出的提示，根据Wyllie算法，将每个item对应的rank计算过程并行开来执行

主要代码如下：

```

start = MPI_Wtime();
MPI_Bcast(qq, N, MPI_INT, 0, MPI_COMM_WORLD);
for (int n = part * my_rank; n < part * (my_rank + 1); n++)
{
    if (qq[n] != -1)
        buf[qq[n]] = n;
    else
        buf[N] = n;
}
MPI_Barrier(MPI_COMM_WORLD);
MPI_Reduce(buf, re, N + 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);
if (my_rank == 0)
{
    for (int n = 0; n < N; n++)
    {
        int hold = re[re[N]];
        re[re[N]] = n;
        re[N] = hold;
    }
}
MPI_Barrier(MPI_COMM_WORLD); /* IMPORTANT */
end = MPI_Wtime();

```

在rank为0的线程输出运行时间

```

if (my_rank == 0)
{
    printf("List ranking mpi time = %f\n", end - start);
    free(qq);
    free(re);
}

```

## 链表求序cuda实现

- `__global__` : 核函数限定符
- `<<< grid, block >>>` : 执行结构参数
- `cudaMalloc` : 在GPU显存中申请空间
- `cudaMemcpy` : 将CPU中的数据copy到GPU中
- `cudaEventRecord`, `cudaEventSynchronize`, `cudaEventRecord`, `cudaEventElapsedTime` : cuda程序计时
- `cudaFree` : 释放空间
- 同样地将进程的MPI并行改为cuda的线程并行即可。

主要代码如下：

```

cudaEvent_t start1;
cudaEventCreate(&start1);
cudaEvent_t stop1;
cudaEventCreate(&stop1);
cudaEventRecord(start1, NULL); // 计时开始

dim3 grid(1, 1);
dim3 block(1, 100);
cudaMemcpy(a, qq, sizeof(int) * N, cudaMemcpyHostToDevice);
my_order<<<grid, block>>>(N, a, b, allnum);
cudaDeviceSynchronize(); // 同步
cudaMemcpy(result, b, sizeof(int) * (N + 1), cudaMemcpyDeviceToHost);
for (int n = 0; n < N; n++)
{
    int hold = result[result[N]];
    result[result[N]] = n;
    result[N] = hold;
}
cudaEventRecord(stop1, NULL);
cudaEventSynchronize(stop1);

float totaltime = 0.0f;
cudaEventElapsedTime(&totaltime, start1, stop1);
printf("List ranking cuda time = %f\n", totaltime);

```

- 核函数

```

__global__ void my_order(int N, int *a, int *b, int allnum)
{
    int part = N / allnum;
    for (int n = part * threadIdx.y; n < part * (threadIdx.y + 1); n++)
    {
        if (a[n] != -1)
            b[a[n]] = n;
        else
            b[N] = n;
    }
}

```

## 稀疏矩阵mpi实现

- MPI\_Init : MPI初始化 , MPI\_Finalize : 结束MPI程序的运行
- MPI\_Comm\_rank : 获取进程的进程号 , MPI\_Comm\_size : 获取进程个数
- MPI\_Send : 发送消息 , MPI\_Recv : 接收消息

- MPI\_Reduce : 归约操作, MPI\_Bcast: 广播, MPI\_barrier: 同步
- 根据老师给出的代码生成稀疏矩阵, 输入输出矩阵都用压缩行存储, 我这里将其重新保存在txt中, 便于我的使用。
- 主要代码如下:  
读取数据并广播:

```
FILE* file;
file = fopen("/public/home/st17341046/read_data.txt", "rb");
while(!feof(file))
{
    struct data c;
    fread(&c,sizeof(struct data),1,file);

    data_A[c.x*N+c.y]=c.data;
    data_B[c.x*N+c.y]=c.data;
}
fclose(file);

MPI_Bcast(matrix_A, N * N, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(matrix_B, N * N, MPI_FLOAT, 0, MPI_COMM_WORLD);
```

进行稀疏矩阵乘法:

```
int a = sqrt(comm_sz);
int sub_N = N / a;

int col, row;
MPI_Comm col_comm, row_comm;
col = my_rank % a;
row = my_rank / a;
MPI_Comm_split(MPI_COMM_WORLD, col, row, &col_comm);
MPI_Comm_split(MPI_COMM_WORLD, row, col, &row_comm);

float *A = (float *)malloc(sizeof(float) * sub_N * sub_N);
float *B = (float *)malloc(sizeof(float) * sub_N * sub_N);
int i, j;
int i_index, j_index;
for (i = 0; i < sub_N; i++)
{
    i_index = row * sub_N + i;
    for (j = 0; j < sub_N; j++)
    {
        j_index = col * sub_N + j;
        A[i * sub_N + j] = matrix_A[i_index * N + j_index];
        B[i * sub_N + j] = matrix_B[i_index * N + j_index];
    }
}
```

```

float *t_A = (float *)malloc(sizeof(float) * sub_N * N);
MPI_Allgather(A, sub_N * sub_N, MPI_FLOAT, t_A, sub_N * sub_N,
MPI_FLOAT, row_comm);

float *t_B = (float *)malloc(sizeof(float) * sub_N * N);
MPI_Allgather(B, sub_N * sub_N, MPI_FLOAT, t_B, sub_N * sub_N,
MPI_FLOAT, col_comm);
float *sub_C = (float *)malloc(sizeof(float) * sub_N * sub_N);
for (i = 0; i < sub_N; i++)
    for (j = 0; j < sub_N; j++)
    {
        sub_C[i * sub_N + j] = 0;
    }
int k, count;
for (count = 0; count < a; count++)
{
    for (i = 0; i < sub_N; i++)
        for (k = 0; k < sub_N; k++)
        {
            if (t_A[count * sub_N * sub_N + i * sub_N +
k] == 0)
                continue;
            for (j = 0; j < sub_N; j++)
            {
                if (t_B[count * sub_N * sub_N + k *
sub_N + j] == 0)
                    continue;
                sub_C[i * sub_N + j] += t_A[count *
sub_N * sub_N + i * sub_N + k] * t_B[count * sub_N * sub_N + k * sub_N +
j];
            }
        }
}
float *C = (float *)malloc(sizeof(float) * N * N);
float *C1 = (float *)malloc(sizeof(float) * N * N);
for (i = 0; i < sub_N; i++)
    for (j = 0; j < sub_N; j++)
    {
        int index = (row * sub_N + i) * N + col * sub_N +
j;
        C[index] = sub_C[i * sub_N + j];
    }
MPI_Reduce(C, C1, N * N, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);

```

输出用压缩行形式存储:

```

FILE *file;
file = fopen("/public/home/st17341056/write_matrix.txt", "wb");
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
    {

```

```

        if (C[i * N + j] == 0)
            continue;
        else
        {
            struct matrix c;
            c.x = i,
            c.y = j;
            c.matrix = C[i * N + j];

            fwrite(&c, sizeof(struct matrix), 1, file);
        }
    }
    fclose(file);

```

## 稀疏矩阵cuda实现

- `__global__` : 核函数限定符
- `<<< grid, block >>>` : 执行结构参数
- `cudaMalloc` : 在GPU显存中申请空间
- `cudaMemcpy` : 将CPU中的数据copy到GPU中
- `cudaEventRecord`, `cudaEventSynchronize`, `cudaEventRecord`, `cudaEventElapsedTime` : cuda程序计时
- `cudaFree` : 释放空间
- 输入输出矩阵读取读出与mpi实现一致
- 计时与之前cuda实现链表求序一致
- 主要实现区别在于核函数

主要代码如下：

```

cudaMallocPitch((void**)&dev_a,&pitch,N*sizeof(float),M);
cudaMemcpy2D(dev_a, pitch, h_a, N * sizeof(float), N * sizeof(float), M,
cudaMemcpyHostToDevice);
cudaMallocPitch((void**)&dev_b,&pitch,N*sizeof(float),M);
cudaMemcpy2D(dev_b, pitch, h_b, N * sizeof(float), N * sizeof(float), M,
cudaMemcpyHostToDevice);
cudaMalloc((void**)&dev_c, M*N*sizeof(float));

dim3 block(block_x,block_y);
dim3 grid(grid_x,grid_y);

...

matrix_multi<<<grid,block>>>(dev_a,dev_b,dev_c,pitch);

```

- 核函数

```
__global__ void matrix_multi(float *A,float *B,float *C,int pitch)
{

    int Row=blockIdx.y*blockDim.y+threadIdx.y;
    int Col=blockIdx.x*blockDim.x+threadIdx.x;

    int thread_id=Row*blockDim.x*gridDim.x+Col;
    int i, j;
    if(thread_id<M)
    {
        for(i=0;i<N;i++)
        {
            float sum=0;
            for(j=0;j<M;j++)
            {
                int h=pitch/sizeof(float);
                float *addr_a=&A[0]+j*h+thread_id;
                float *addr_b=&B[0]+j*h+i;
                sum+=(*addr_a)*(*addr_b);
            }
            C[thread_id*N+i]=sum;
        }
    }
}
```

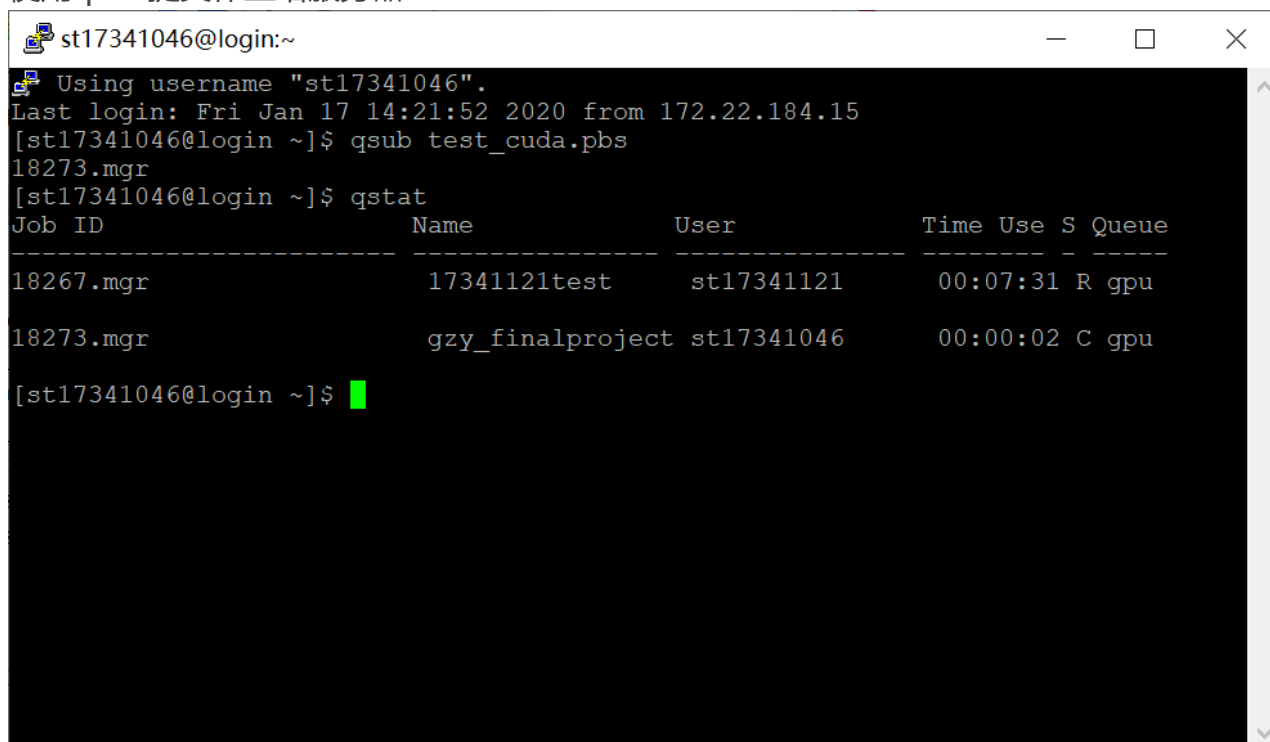
## 实验过程

1. 在本机中vscode, mpi, cuda环境中编写好代码。
  - 编译时使用命令

```
source /public/software/profile.d/mpi_openmpi-intel-2.1.2.sh
mpic++ -g gzy_ex.cpp -o gzy_ex
source /public/software/profile.d/cuda10.0.sh
nvcc -g gzy_ex.cu -o gzy_ex
```

2. 利用winscp上传至集群中
3. 打开putty命令行界面, 编写test.pbs,使用qsub等命令将作业提交给PBS服务器, 进行运行, 在自己的文件夹中查看结果, 并记录。  
具体操作见下图:

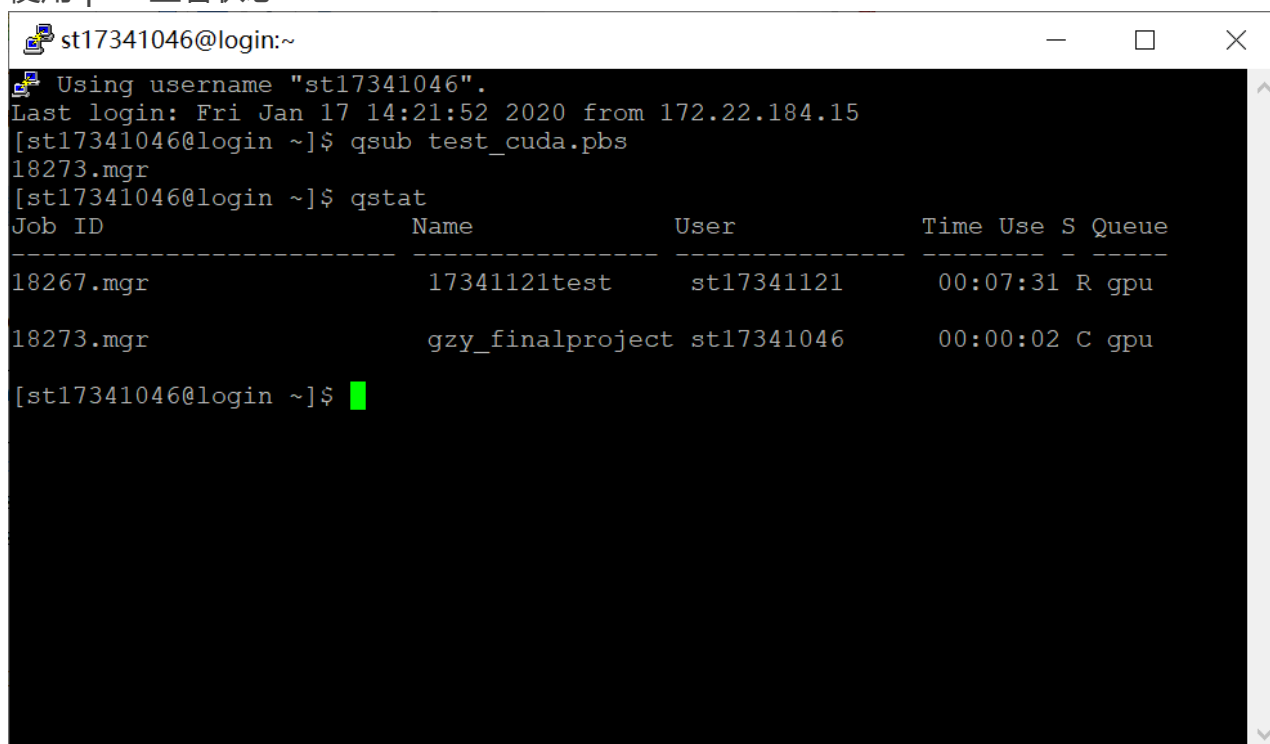
- 使用qsub提交作业给服务器



A terminal window titled "st17341046@login:~" showing the execution of qsub and qstat commands. The output of qsub shows a job named "test\_cuda.pbs" with ID 18273.mgr. The output of qstat shows two jobs: 18267.mgr (Name: 17341121test, User: st17341121, Time Use: 00:07:31, S: R, Queue: gpu) and 18273.mgr (Name: gzy\_finalproject, User: st17341046, Time Use: 00:00:02, S: C, Queue: gpu).

```
st17341046@login:~  
Using username "st17341046".  
Last login: Fri Jan 17 14:21:52 2020 from 172.22.184.15  
[st17341046@login ~]$ qsub test_cuda.pbs  
18273.mgr  
[st17341046@login ~]$ qstat  
Job ID              Name                User                Time Use S Queue  
-----  
18267.mgr           17341121test        st17341121          00:07:31 R gpu  
18273.mgr           gzy_finalproject    st17341046          00:00:02 C gpu  
[st17341046@login ~]$
```

- 使用qstat查看状态




A terminal window titled "st17341046@login:~" showing the execution of qstat. The output is identical to the previous terminal window, displaying the status of two jobs: 18267.mgr and 18273.mgr.

```
st17341046@login:~  
Using username "st17341046".  
Last login: Fri Jan 17 14:21:52 2020 from 172.22.184.15  
[st17341046@login ~]$ qsub test_cuda.pbs  
18273.mgr  
[st17341046@login ~]$ qstat  
Job ID              Name                User                Time Use S Queue  
-----  
18267.mgr           17341121test        st17341121          00:07:31 R gpu  
18273.mgr           gzy_finalproject    st17341046          00:00:02 C gpu  
[st17341046@login ~]$
```

- 编写test.pbs脚本

下图为一个mpi的pbs脚本示例:




 /public/home/st17341046/test\_mpi.pbs - st17341046@222.200.180.115 - 编辑器 - WinSCP

```
#PBS -N gzy_finalproject
#PBS -l nodes=16:ppn=32
#PBS -j oe

echo "This is gzy's lab"$PBS_JOBID@PBS_QUEUE
cd $PBS_O_WORKDIR
source /public/software/profile.d/mpi_openmpi-intel-2.1.2.sh
mpic++ matmul_mpi.cpp -o matmul_mpi
mpiexec -np 4 ./matmul_mpi
```

下图为一个cuda的pbs脚本示例：

 /public/home/st17341046/test\_cuda.pbs - st17341046@222.200.180.115 - 编辑器 - WinSCP

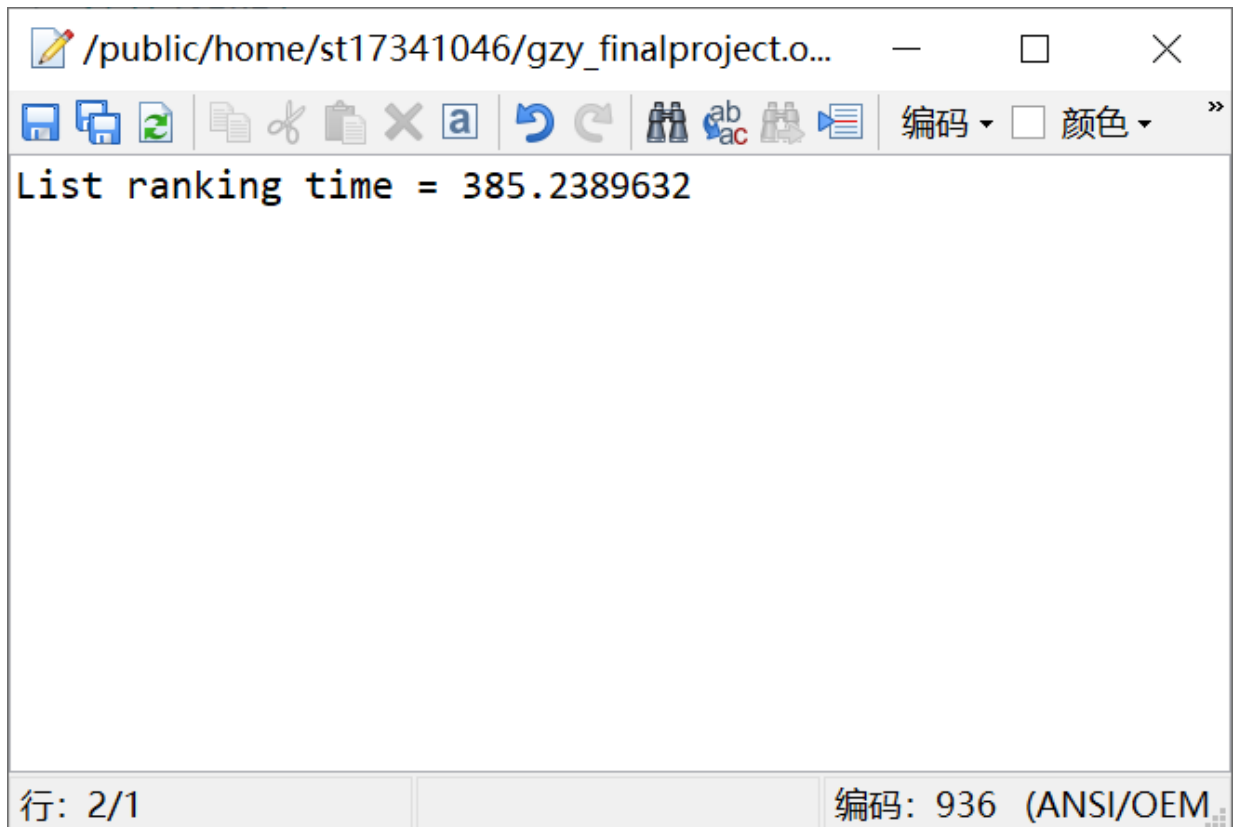
```
#PBS -N gzy_finalproject
#PBS -l nodes=1:ppn=32:gpus=1
#PBS -j oe
#PBS -q gpu

source /public/software/profile.d/cuda10.0.sh
nvcc listranking_cuda.cu -o listranking_cuda
./listranking_cuda
```

- 得到输出文件
- 打开文件夹中的输出文件并记录数据。  
要得到可靠的运行时间数据只需要qsub多次求平均和即可。

## 实验结果及所得结论

- 链表求序
  - 串行时间  
串行时间不妨以单线程时间代替。



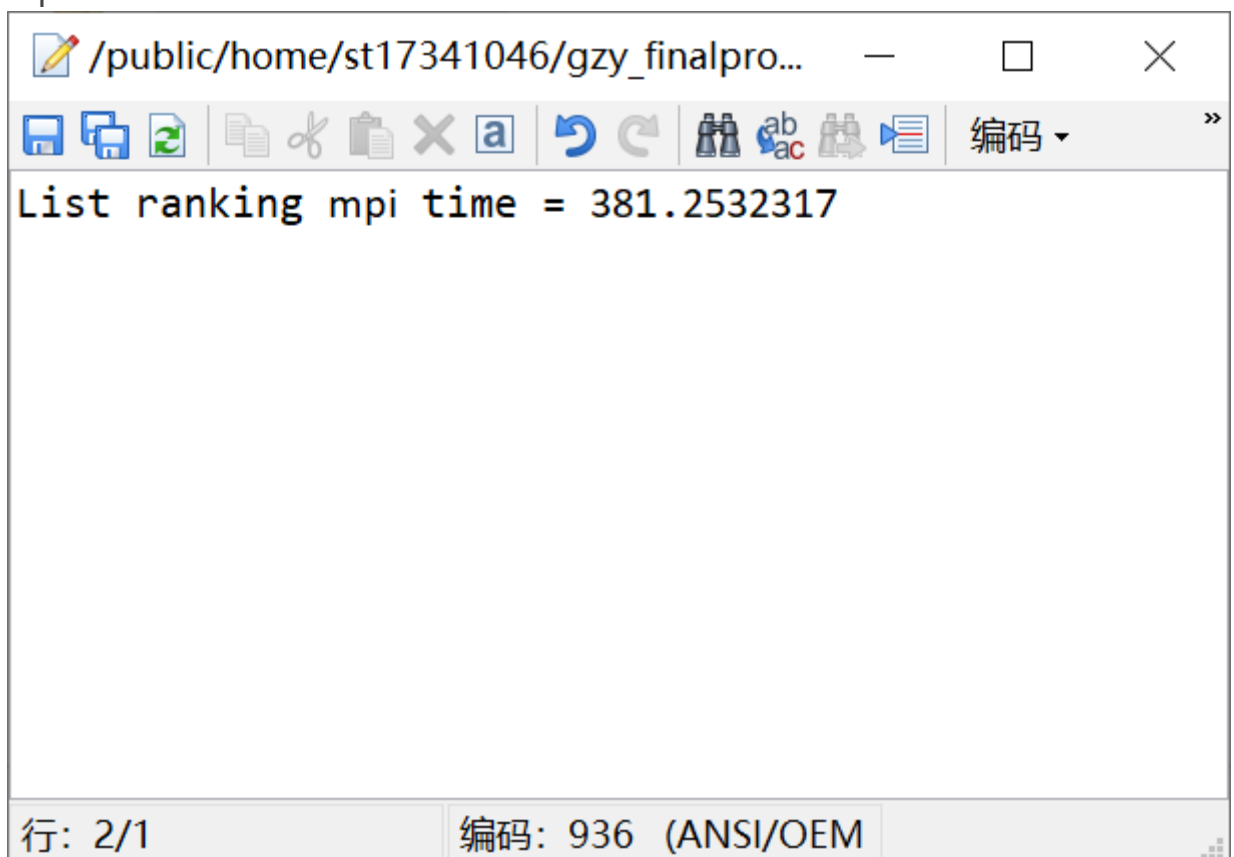
/public/home/st17341046/gzy\_finalproject.o... — □ ×

保存 复制 粘贴 剪切 删除 撤销 重做 查找 替换 编码 颜色 »

```
List ranking time = 385.2389632
```

行: 2/1 编码: 936 (ANSI/OEM)

○ mpi



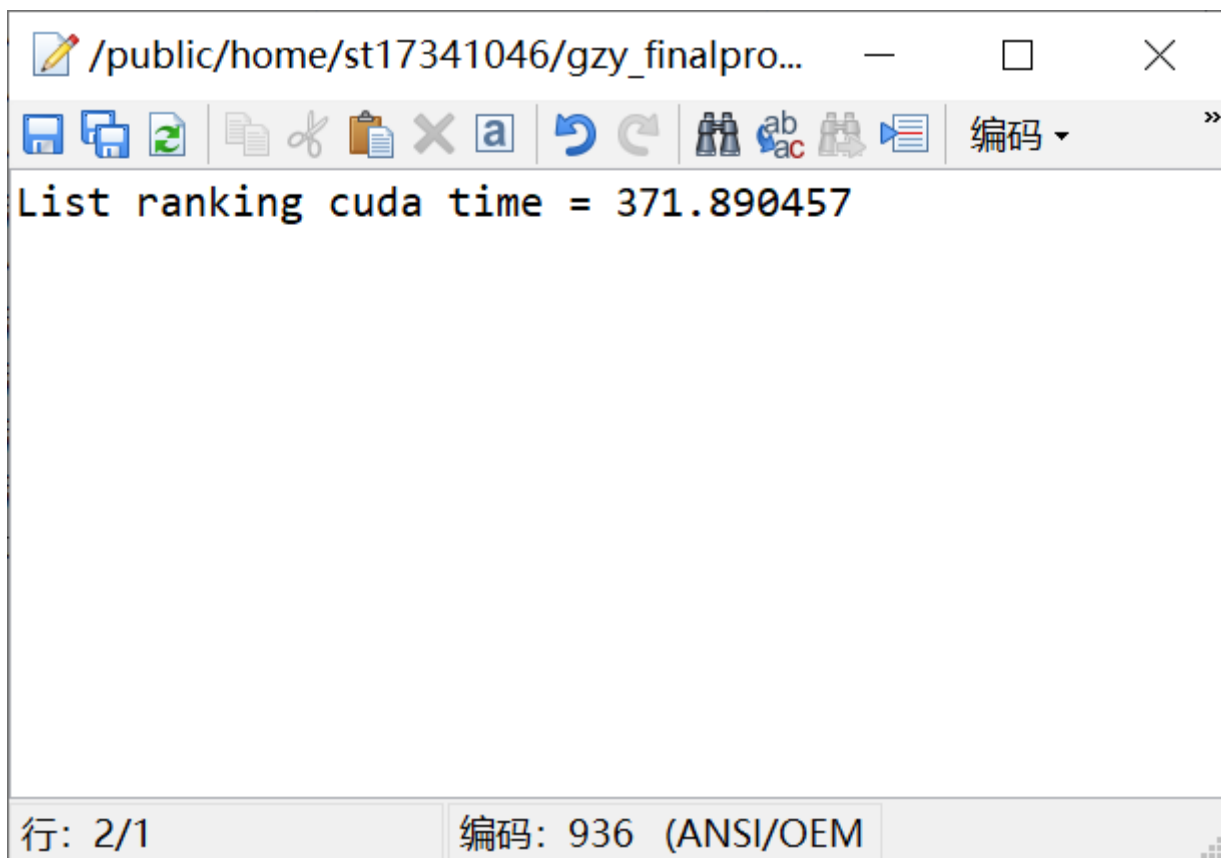
/public/home/st17341046/gzy\_finalpro... — □ ×

保存 复制 粘贴 剪切 删除 撤销 重做 查找 替换 编码 »

```
List ranking mpi time = 381.2532317
```

行: 2/1 编码: 936 (ANSI/OEM)

- cuda



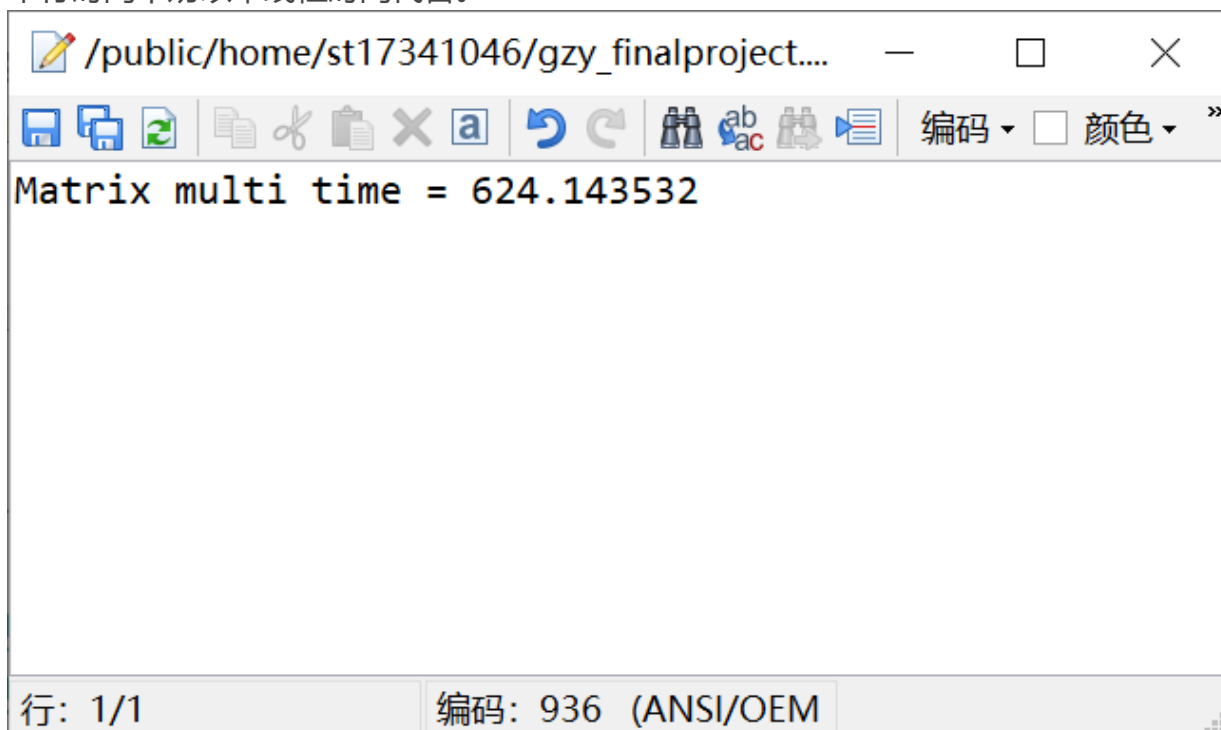
```
/public/home/st17341046/gzy_finalpro...  
List ranking cuda time = 371.890457  
行: 2/1 编码: 936 (ANSI/OEM)
```

可以看到，链表求序的并行实现对串行实现的改进效果并不明显甚至还可能比之更差劲。

- 稀疏矩阵

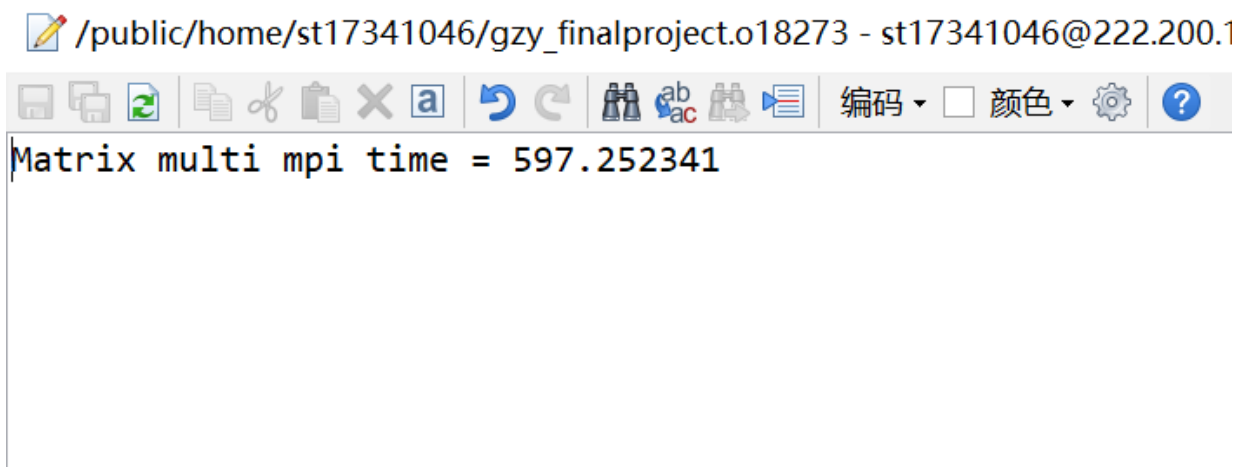
- 串行时间

串行时间不妨以单线程时间代替。



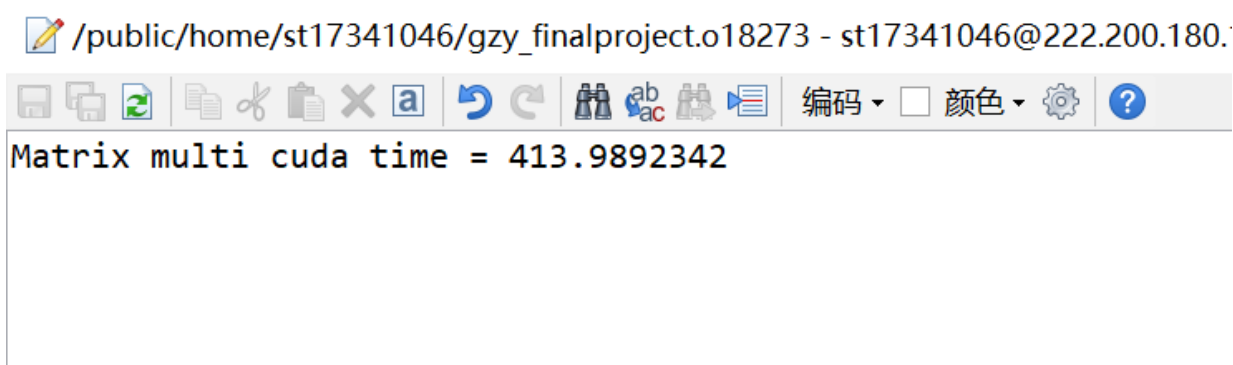
```
/public/home/st17341046/gzy_finalproject....  
Matrix multi time = 624.143532  
行: 1/1 编码: 936 (ANSI/OEM)
```

- mpi



A terminal window with a title bar showing the path `/public/home/st17341046/gzy_finalproject.o18273 - st17341046@222.200.1`. The terminal has a standard Linux-style toolbar with icons for file operations and editing. The command prompt shows `Matrix multi mpi time = 597.252341`.

- cuda



A terminal window with a title bar showing the path `/public/home/st17341046/gzy_finalproject.o18273 - st17341046@222.200.180.`. The terminal has a standard Linux-style toolbar. The command prompt shows `Matrix multi cuda time = 413.9892342`.

可以看到有少许改进。

## 所遇问题及解决方案

- 问题理解不明确

一开始老师给出的问题描述十分模糊，大部分同学都难以理解其含义，究竟要做什么。好在临近截止日期时，终于给出了关于实验1的提示，在阅读了其中的几个算法以及究竟要做什么后，才匆忙地开始实验，所以这一次做的有些仓促，实验报告也没有写的十分详细。

- mpi, cuda混合编程问题

这次实验用到了mpi和cuda实现并比较时间效率。关于不同语言的编程，在同一次实验中用到，mpi编程也有一段时间没用到了，难免会有些生疏，所以在编程过程中也是多次地去查阅函数才完成了实验。也算是对之前的编程的一次复习与检验。

## 心得体会

这一次的实验实现要求较多，实验时间较少，一开始给的要求也不明确，造成了这次实验弄的比较混乱，尽管如此，经过最后这几天在家中的编程，还是完成了。对之前mpi, cuda编程的使用都是一次很好的复习巩固，对于链表求序的算法和稀疏矩阵的乘法都有了进一步自己的思考，时间关系就不在这里——细说。总而言之，这一次的实验虽然麻烦了一些，但也令我收获良多，受益匪浅。

