**K-Nearest Neighbor Problem实验报告**

黄 欢

（计算机科学与技术系 计33 2013011331）

1. **实验要求与内容**

Problem Description

In this assignment, you need to implement a program to solve the k-nearest neighbor problem in CUDA. The k-nearest neighbor problem is described as follows. Given m nodes in ndimension, for each node, output its k-nearest neighbors. The distance is defined using Euclidean distance. For example, for node v at position (v1, v2, …, vn) and node u at position (u1, u2, …, un), their distance is defined as d(u,v) =[ sum\_{i=1,..n} (v\_i-u\_i)^2]^{1/2}.

Input

An nxm matrix of integers and a number k, where n is in [1,1000], m is in [10, 1000], k is in [1,10], and the integer elements are in [-5,5]. The number n, m, and the position matrix are input from a file.

m n k

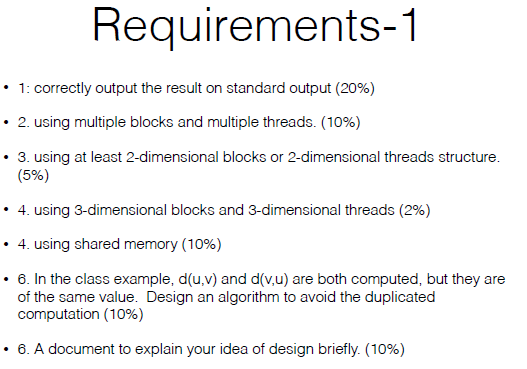


Output

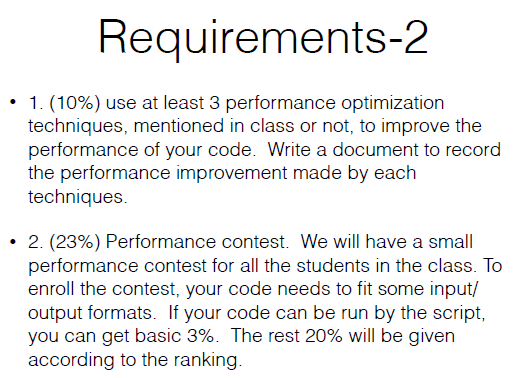
for each node, output the indices of nodes of its k nearest neighbors.

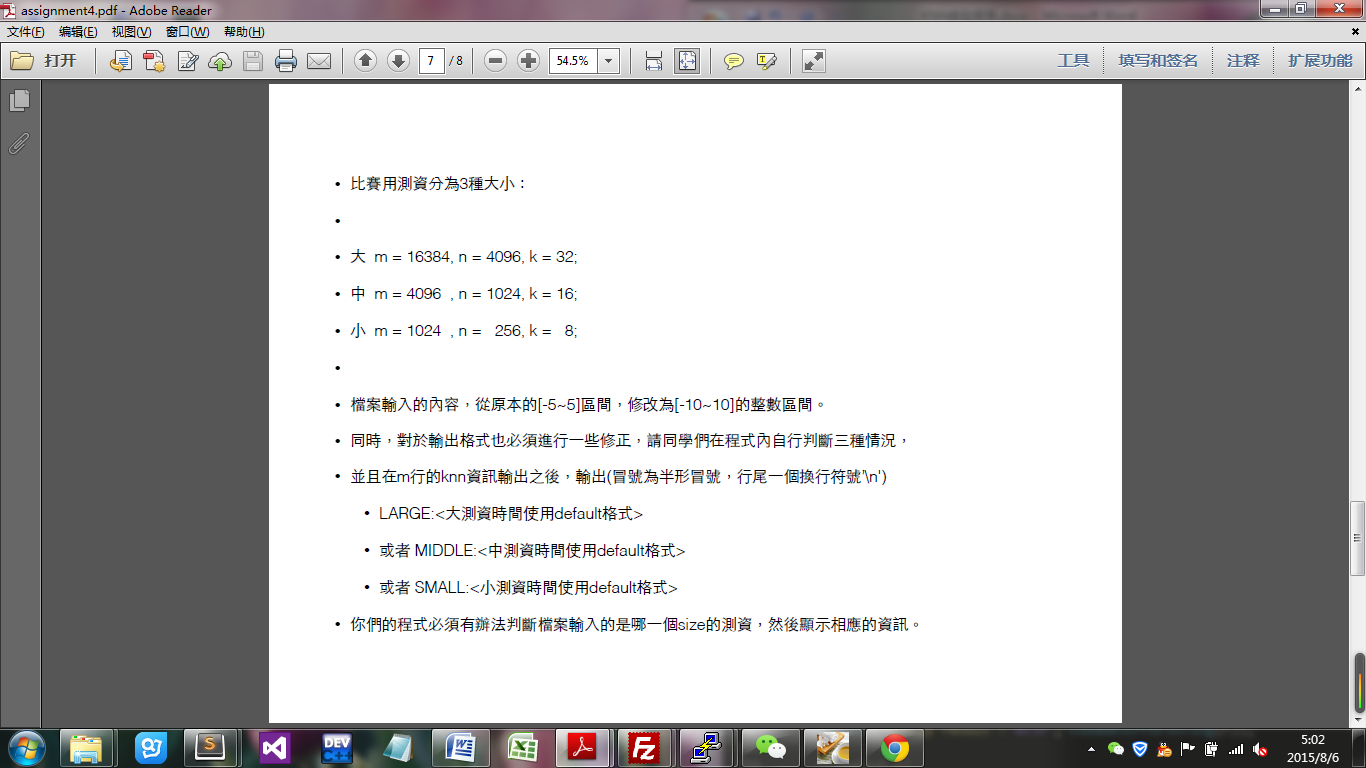


Requirements-1



Requirements-2





1. **实验方法与实现**
2. **Sequential Code**

//求出所有点两两之间的距离（避免重复计算）

for (int i = 0; i < m; i++) {

mul[i][i] = MAX\_VALUE;

for (int j = i+1; j < m; j++) {

mul[i][j] = getSum(i, j);

mul[j][i] = mul[i][j];

}

}

//求k个最近点

for (int i = 0; i < m; i++) {

getknn(i);

}

1. **Sequential求距离**

Sequential求Euclidean distance，方法如下：

int getSum(int i, int j) {

int tmpSum = 0;

for (int g = 0; g < n; g++) {

tmpSum += ((element[i][g] - element[j][g]) \* (element[i][g] - element[j][g]));

}

return tmpSum;

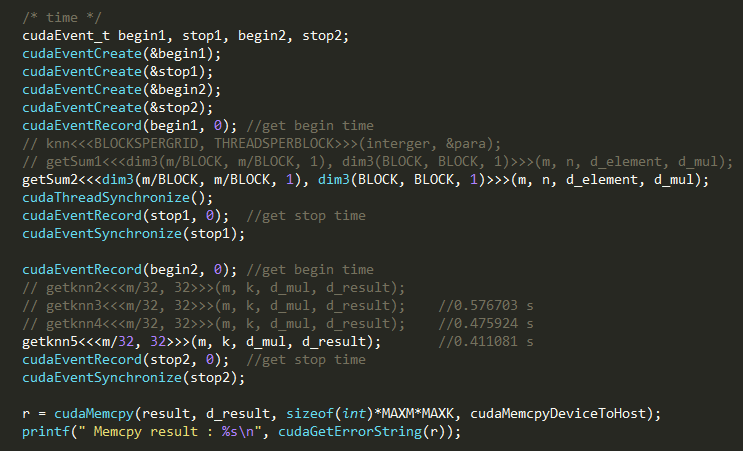
}

1. **Sequential求knn**

采用插入排序，由于k比较小，从最后一个开始比较较快。

1. **Cuda思想**

解决本题的时间主要消耗在求距离上。我们可以采用multiple blocks、multiple threads、多维（dim3）、shared memory、避免重复计算d(u, v)和d(v, u)等方法来并行加速。具体如下：

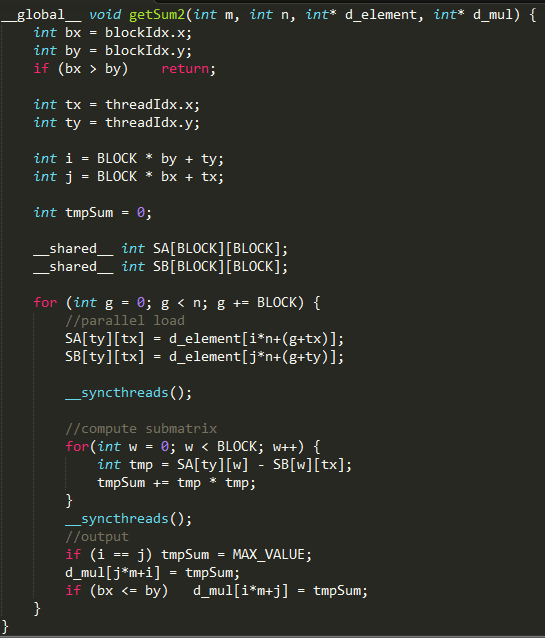


求距离采用的是二维的grid和block，每一个thread计算一对点的距离。

由于从global memory中读取的时间大于把数据拷贝到shared memory再读取的时间，所以通过使用shared memory可以节约时间。

由于d(u, v)和d(v, u)的值相等，所以可以通过“先return，再赋值“，来避免重复运算，从而节省时间。

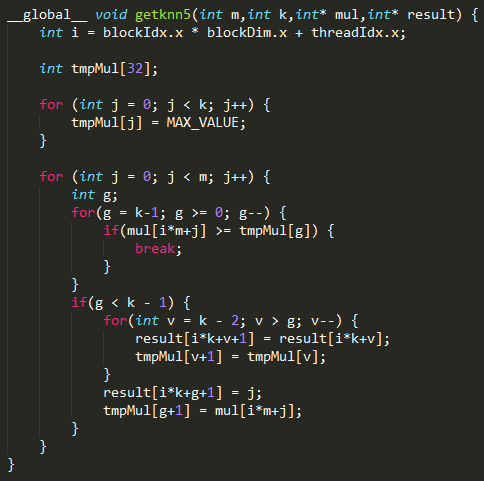
具体代码如下：



求k个最小点采用的是一维的grid和block，每一个thread计算一个点的K个最近点。

内部实现依旧采用Sequential的方法，即插入排序，从最大值开始比较。

具体代码如下：



1. **Cuda思想**

设置合适的BlockPerGrid和ThreadPerBlock，每一个thread都做同样的较为简单的运算，利用总thread数的优势，来达到高速并行的目的，减少运算的时间。

1. **相关变量**

|  |  |
| --- | --- |
| 变量名 | 意义 |
| m | **点数** |
| n | **每个点的维度** |
| k | **K个最近点** |
| element | **点坐标（二维）** |
| mul | **点距离（二维）** |
| result | **最终结果（二维）** |
| getSum | **求距离函数** |
| getknn | **求knn函数** |

1. **时间**

在pthread中，运用clock\_gettime求时间，代码如下：

#include <time.h> //-lrt

//time

struct timespec time1 = {0, 0};

struct timespec time2 = {0, 0};

clock\_gettime(CLOCK\_MONOTONIC, &time1);

//do something

clock\_gettime(CLOCK\_MONOTONIC, &time2);

printf(" CLOCK\_TIME: %lf ms\n", (time2.tv\_sec - time1.tv\_sec) \* 1000 + (time2.tv\_nsec - time1.tv\_nsec) / 1e6);

在cuda中，运用cudaEvent求时间，代码如下：

#include <cuda\_runtime.h>

//time

cudaEvent\_t begin, stop;

cudaEventCreate(&begin);

cudaEventCreate(&stop);

//Get begin time

cudaEventRecord(begin, 0);

//do something on GPU

getSum2<<<dim3(m/BLOCK, m/BLOCK, 1), dim3(BLOCK, BLOCK, 1)>>>(m, n, d\_element, d\_mul);

cudaThreadSynchronize();

getknn5<<<m/32, 32>>>(m, k, d\_mul, d\_result);

//Get stop time

cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);

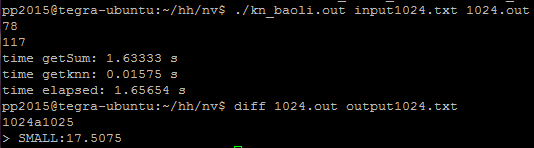
float elapsedTime;

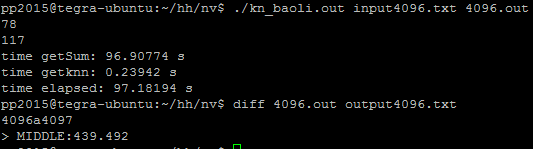
cudaEventElapsedTime(&elapsedTime, begin, stop);

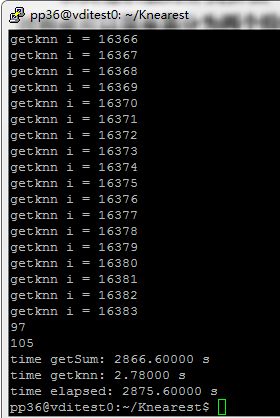
printf(" Execution Time on GPU: %3.20f s\n", elapsedTime/1000);

1. **Performance**
2. **运行时间及对比**

串行large所需时间将近48分钟，由下图（m= 16384）可知大部分时间花在了计算点对之间的距离上，而接下来我们会看到并行程序发挥的巨大作用。







串行时间数据整理如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 时间/s | 求距离 | 求knn | 总计 |
| Text 1024 | 1.63333 | 0.01575 | 1.65654 |
| Text 4096 | 96.90774 | 0.23942 | 97.18194 |
| Text 16384 | 2866.6 | 2.78 | 2875.6 |

并行时间数据整理如下表：

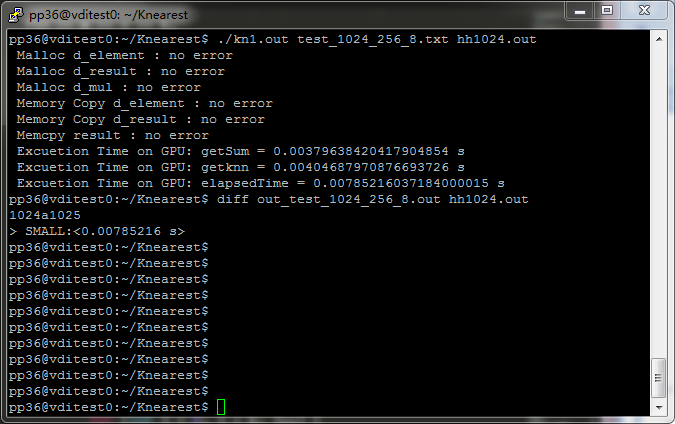
|  |  |  |  |
| --- | --- | --- | --- |
| 时间/s | 求距离 | 求knn | 总计 |
| Text 1024 | 0.003796 | 0.004046 | 0.007852 |
| Text 4096 | 0.231635 | 0.025109 | 0.256753 |
| Text 16384 | 14.722307 | 0.410728 | 15.133045 |

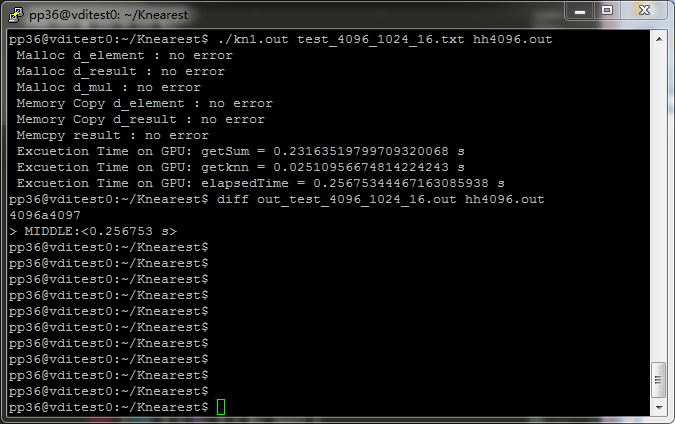
由图得，加速主要体现在求距离的部分，而且从时间的倍数来看，中等数据的加速效果最好。

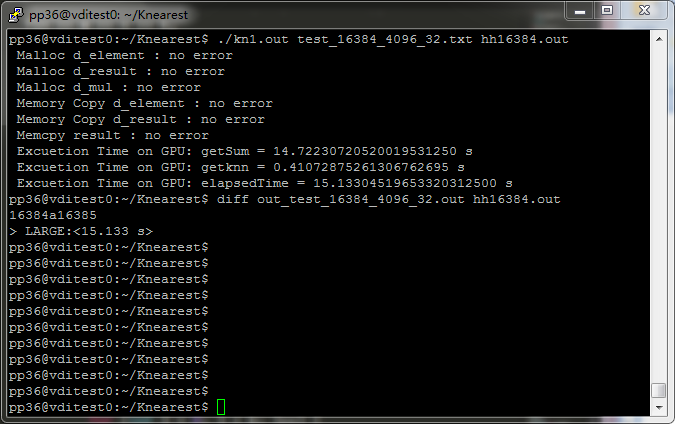
由表中数据可知，随着m数目的增加和n维度的增加，各项活动的时间都有增加，符合常理。而与串行相比，并行的计算时间明显减少。

由于运行所需的时间会受到环境因素的影响，每一次的运行结果都不一样，但是相差也不会很多，所以表中数据仍有一定参考价值。

截图如下：







1. **实验实现总结与心得体会**

通过K-Nearest Neighbor Problem实验，我学习并掌握了利用多维并行实现程序，也钻研了KNN的一些实现方法。这个实验很有趣，同时要在短时间内完成，也很考验我们。在不断改进自己的算法和调程序的过程中，我学会了很多知识，提升了代码能力，收获很大。并行程式在实际生活中有着广泛应用，为我们节省了大量时间，相信随着科技的发展，并行程序设计将有更大的突破。

最后，非常感谢老师、助教的辛勤指导，我会继续探索、继续努力！