同济大学 TONGJI UNIVERSITY

系统编程并行编程课程设计报告

Project Report for System Programming and Parallel Programming

基于 CUDA 和 TCP 通信的大数据双机加速计算

Big Data Dual-Computer Acceleration Calculation Based on CUDA and TCP Communication

学　　院	电子与信息工程学院		
专　　业	控制科学与工程		
学生姓名	黄定梁	学　　号	2130724
学生姓名	陈俊柱	学　　号	2130723
指导教师	王晓年		

2022 年 1 月 9 日

同济大学系统编程并行编程课程设计项目报告

Team Project Report

<基于 CUDA 和 TCP 通信的大数据双机加速计算>

<2022.01.09>

目录

1 环境/技术简介

1.1 程序运行环境

1) server 端计算机
操作系统：Ubuntu 18.04.5 LTS
运行环境：VSCode 或 Bash 终端
2) client 端计算机
操作系统：Ubuntu 16.04 LTS
运行环境：VSCode 或 Bash 终端

1.2 硬件配置

1) server 端计算机
CPU：Intel Core™ i7-8700K CPU @ 3.70GHz×12
GPU：NVIDIA TITAN Xp COLLECTORS EDITION
2) client 端计算机
CPU：Intel Core™ i7-8700K CPU @ 3.70GHz×12
GPU：NVIDIA GeForce RTX 2080 Ti

1.3 技术简介

1) CUDA

CUDA（Compute Unified Device Architecture），是显卡厂商 NVIDIA 推出的运算平台。CUDA™是一种由 NVIDIA 推出的通用并行计算架构，该架构使 GPU 能够解决复杂的计算问题。它包含了 CUDA 指令集架构（ISA）以及 GPU 内部的并行计算引擎。开发人员可以使用 C 语言、C++等高级语言来为 CUDA™架构编写程序，所编写出的程序可以在支持 CUDA ™的处理器上以超高性能运行。GPU 包含了比 CPU 更多的处理单元，更大的带宽从而以多核并行的优势加速计算。

2) TCP

传输控制协议（TCP，Transmission Control Protocol）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 定义。TCP 是因特网中的传输层协议，使用三次握手协议建立连接。当主动方发出 SYN 连接请求后，等待对方回答 SYN+ACK，并最终对对方的 SYN 执行 ACK 确认。这种建立连接的方法可以防止产生错误的连接，TCP 使用的流量控制协议是可变大小的滑动窗口协议。

2 基础模块描述

2.1 宏定义说明

1) #define MAX_THREADS

单机 CPU 可真实调用的最大线程数，定义为 64，在此程序中用于后续数据量的计算。

2) #define SUBDATANUM

单机 CPU 每个线程负责处理的数据量，在单机运行时定义为 2000000，在双机加速运行时分别在两台计算机上定义为 1000000，在此程序中用于后续数据量的计算。

3) #define DATANUM

单机负责处理的数据总量，其值定义为(SUBDATANUM * MAX_THREADS)。

4) #define BLOCKSIZE

用于传给双机加速程序中排序功能所调用的 CUDA 核函数，用于规定核函数声明的 GPU 的线程块（Block）数量。经测试，在 DATANUM 的数据量下，其被定义为 4096，从而达到最高加速比。

2.2 基础功能设计

2.2.1 串行函数 sumCommon()

以串行的方式使用 for 循环进行累加，累加的数值为每一位数据进行过 log(sqrt())操作后的数值。当以 float 作为数据类型进行累加操作时，发现运算结果与理论值相差较远，因此改用 double 作为运算的数据类型，该函数也返回 double 型的累加结果。该求和算法的时间复杂度为 O(n)。

2.2.2 串行函数 maxCommon()

以串行的方式使用 for 循环进行逐位比较，进行比较的数值均为经过 log(sqrt())操作后的数值。将逐次比较后取得的 float 型最大值作为返回值返回。该求最大值算法的时间复杂度为 O(n)。

2.2.3 串行函数 sortCommon()

采用归并排序算法进行排序，其时间复杂度为 $O(n\log_2 n)$，对于本题中的大量顺序排列的原始数据，常用的冒泡排序、快速排序等排序算法的时间复杂度均为 $O(n^2)$，目前普通计算机所使用的 CPU 无法在有效时间内完成该排序任务，其加速比可认为是“无穷大”，故选择归并排序。归并排序（MERGE-SORT）是利用归并的思想实现的排序方法，该算法采用经典的分治（divide-and-conquer）策略（分治法将问题分(divide)成一些小的问题然后迭代求解，而治(conquer)的阶段则将分的阶段得到的各答案“修补”在一起，即分而治之）。归并排序的算法基本思想如下图所示：

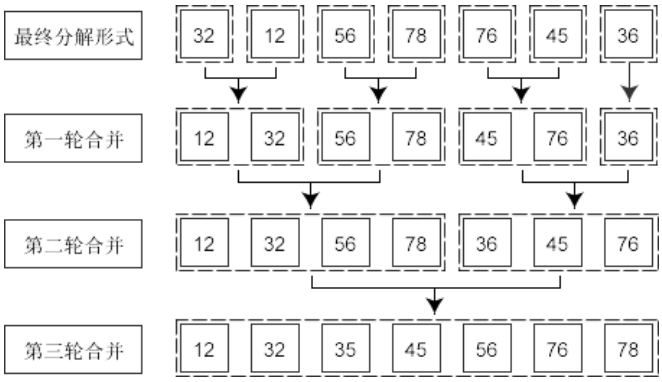

图 2.1 归并排序（MergeSort）基本思想

3 加速设计

3.1 SpeedUp 类

SpeedUp 类封装有与 CUDA 加速相关的各种方法，包括类的构造和析构、数据初始化、CUDA 初始化、打印 CUDA 信息、CUDA 加速版求和函数、CUDA 加速版求最大值函数、CUDA 加速版归并排序函数和排序结果校验函数。

3.3.1 SpeedUp 类构造与析构

在类的构造函数中，完成了对 CUDA 的初始化操作，在终端打印 CUDA 硬件的信息，

并创建了 CUDA 消息处理器，申请了原始数据所需的 CPU 内存空间，还对其中的数据进行了初始化。在析构函数中，释放了创建对象时开辟的 CPU 内存空间，销毁了 CUDA 的消息处理器，并且重置了 GPU，为之后程序的正常运行提供了保障。数据初始化的顺序为由小到大，即从 1 初始化至 DATANUM；排序结果的校验则验证排序后的数据是否为倒序，即从 DATANUM 至 1。SpeedUp 类的构造与析构函数的编写降低了用户使用该类进行基础运算 CUDA 加速的门槛，用户仅需创建一个 SpeedUp 的对象，即可直接调用其中的加速函数。

1) 不可分页数据

在后续的 CUDA 加速操作中，需要将 CPU 内存中的数据拷贝至 GPU 内存中，此过程将带来较大的开销，因此考虑在存储原始数据时就进行优化。即将原始数据存储为不可分页数据，使其始终存在于物理内存中，不会被分配到低速的虚拟内存中。使用 cudaMallocHost() 函数开辟 CPU 内存空间即可实现该目的，其实质是强制让系统在物理内存中完成内存申请和释放的工作，不参与页交换，从而提高系统效率，其原理如图 3.1 所示。

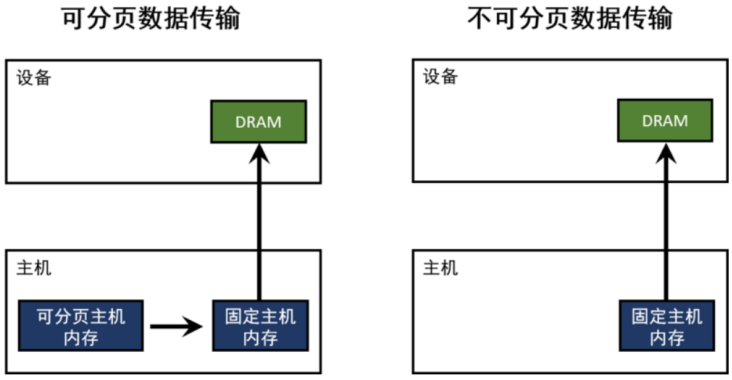

图 3.1 不可分页数据加速原理

3.3.2 成员函数 sumSpeedUp()

求和函数的加速思路整体基于并行归约算法，将数据分组进行层层并行运算，最后再统一归约。该函数内部使用 CUDA 进行加速的过程分为七步，程序流程图如图 3.2 所示。值得一提的是，在步骤一中配置核函数最优参数的方法是通过调用 CUDA 库中的 cudaOccupancyMaxPotentialBlockSize()函数来计算最优的线程块数量和块中的线程数，在题设数据量下，其最优配置为：线程块数量=62500，块中线程数=1024。步骤六中为串行执行的 for 循环对由 GPU 上每个线程块返回的数据进行收割归约，因开辟线程开销占比较大，因而并未采取多线程并行。以下将对该函数中所采取的加速方法进行进一步的说明。

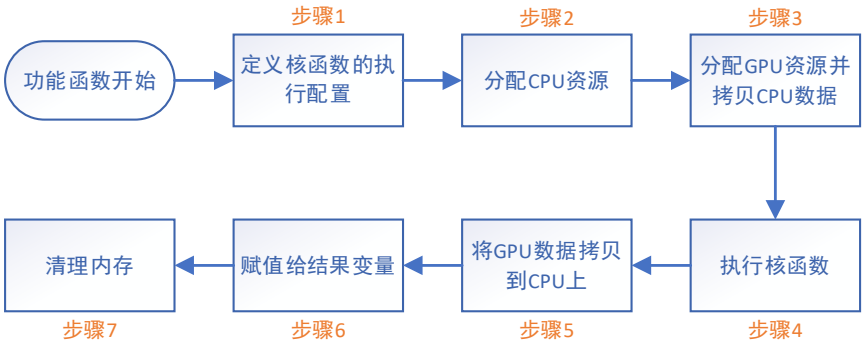

图 3.2 sumSpeedUp()函数程序流程图

1) 并行归约

由于加法的交换律和结合律，数组可以以任意顺序求和。首先把输入数组划分为更小的

数据块，之后用一个线程计算一个数据块的部分和，最后把所有部分和再求和得出最终结果。计算时首先将相邻的两数相加，结果写入第一个数的存储空间内。第二轮迭代时我们再将第一次的结果两两相加得出下一级结果，一直重复这个过程最后直到我们得到最终的结果，而其余单元里面存储的内容是我们不需要的。这个过程中每一轮迭代后，选取被加数的跨度将翻倍。该算法将时间复杂度由 O(n)变为了 $O(\log_2 n)$。在每一层的 CUDA 加速运算中，又由多个线程分别负责一组数据的运算，因此实现了大量线程的并行运算，极大提升了速度。该并行求和算法原理图如图 3.3 所示。

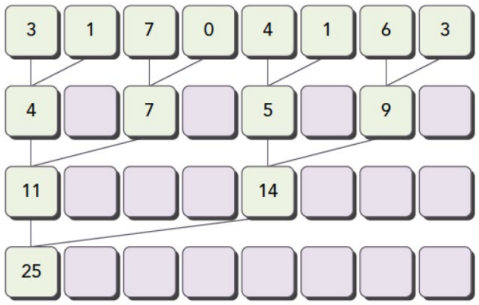

图 3.3 并行归约（相邻）求和算法原理图

2) 交错配对归约

与前述的相邻并行归约不同，交错配对归约改变了选取数据的跨度，始终对位于前部的内存单元进行修改。该方法使前部的线程束最大程度地利用数据，将内存请求集中于该部分活跃的线程束，而后部的线程束虽同时工作但不请求内存，该方法使其最大效率地利用带宽。

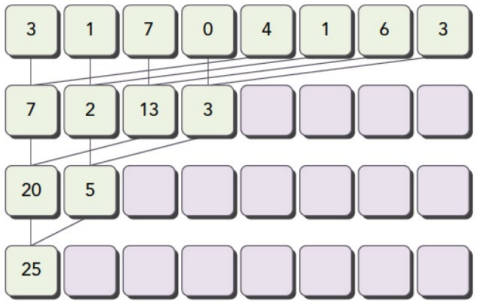

图 3.4 并行归约（交错）求和算法原理图

3) 共享内存

核函数是 CUDA 加速计算的核心，承担了计算部分的主要功能，因此对内存的请求主要集中于核函数中。为减少对全局内存访问的开销，此处引入共享内存作为可编程管理缓存，让其缓存片上的数据，从而减少核函数中全局内存访问的次数。

3.3.3 成员函数 maxSpeedUp()

求最大值函数的加速思路与前述求和函数大致相同，除运算方式不同外没有本质上的变化，其整体基于并行归约算法。其程序流程图如图 3.2 所示。在题设数据量下，其核函数最优配置为：线程块数量=62500，块中线程数=1024。其采取的加速方法与求和函数中的一致，可参考上一小节的内容。

3.3.4 成员函数 sortSpeedUp()

排序函数的加速思路与前述两个函数有所不同，由于其不仅设计存储单元数据之间的运算，还涉及到其存储位置的调配，因此考虑其他加速思路。其程序流程图仍如图 3.2 所示。在步骤一中配置核函数的基础参数，其线程块数量由宏定义的 BLOCKSIZE 确定，而每个线程块中仅包含一个线程，因此将其核函数配置为：线程块数量=4096，块中线程数=1。在单

机未加速程序中已经使用归并排序算法将其运算时间压缩到了较低水平，因此 CUDA 加速方面较难从算法本身对其进行进一步的加速，在该部分实现中的加速策略如图 3.5 所示，以下对其采取的加速方法进行进一步的说明。

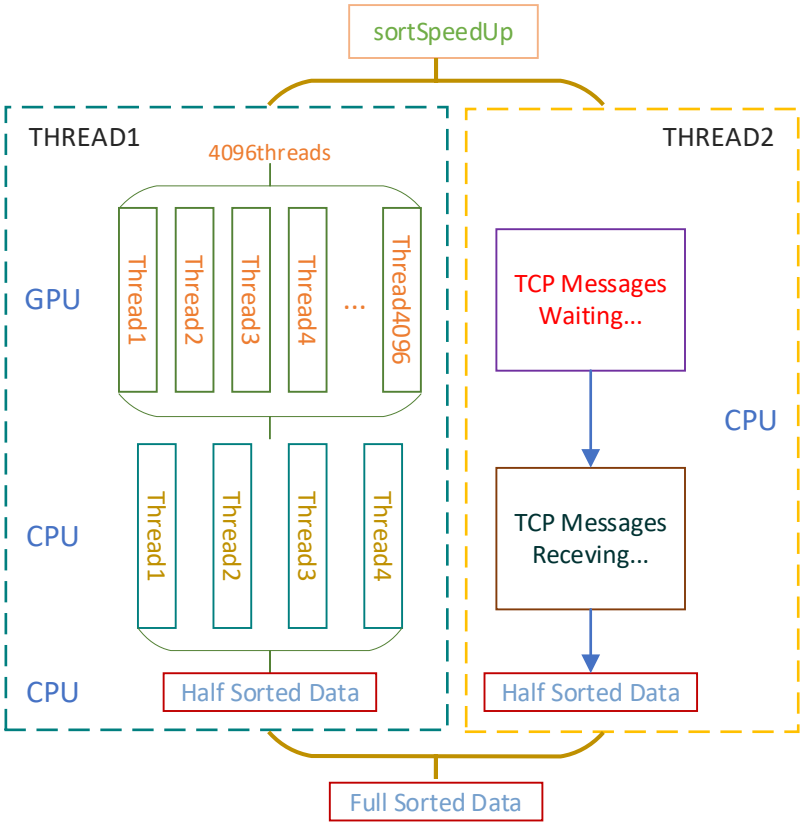

图 3.5 排序功能加速策略架构图

1) CUDA 多线程并行

使用 GPU 中的 4096*1 个线程对归并排序迭代过程中的每一组数据分别排序，每个线程分别负责 DATANUM/BLOCKSIZE 个数据的排序运算，达到 4096 个线程的并行排序，从而提升归并排序的运算速度。此加速方法为 CUDA 最基础的加速方法，前述函数也在使用。

2) CPU 多线程收割

当 CUDA 完成 4096 组数据分别的组内排序之后，需要将数据重新拷贝回 CPU 内存中，此时对 4096 组数据进行统一的归并排序仍然具有巨大的计算量。因此此时在 CPU 上开辟 4 个线程同时对 4096 组数据进行收割，每个线程负责收割 1024 组数据。此轮收割完成后再对 4 组有序的数据进行一轮收割，即可完成此台计算机负责的所有数据的排序工作。

3) 运算与 TCP 接收双线程同步

由于需要进行双机协同运算，双机由于硬件型号和性能的个体差异，可能不能同时完成各自数据的运算。在双机协同大数据运算的实现中发现，局域网内双机 TCP 通信传输大量数据需要极大的时间开销，因此将接收数据与本机运算并行起来成为加速的考虑重点之一。作为 server 的计算机运算速度相对 client 计算机较慢，若需要 server 计算机完成本机运算后才能监听 TCP 消息并接收数据将造成较大的时间开销。因此在 server 端的主函数中开辟双线程，线程 1 负责进行本机数据的排序运算，线程 2 负责监听 TCP 信道并接收 client 发送来的有序数据。

4 通信设计

4.1 tcp_send 类

1）类内属性：类内属性包括公有属性和私有属性。其中，公有属性有 TCP 通信的 socket 号，私有属性有通信地址和通信初始化时的发送接收缓存区。类属性具体命名如下：

```
int socket_fd;          //socket 号
struct sockaddr_in addr;            //通信地址
char buffer[255];           //接收缓存区
double hot;                 //发送缓存区
```

2）类内函数：类内函数包括构造函数、析构函数、发送初始化化函数、客户端初始化函数、发送浮点型数组函数、发送单个浮点数函数和发送单个双精度数函数。其中，发送初始化函数完成与另一台计算机的通信连接，构造函数调用了发送初始化函数，完成发送端的初始化工作，析构函数执行关闭 TCP 通信端口的操作。类函数具体如下：

```
tcp_send()                  //构造函数
{
    send_initial();             //调用类内发送初始化函数
}
~tcp_send()                 //析构函数
{
    close(fd);                  //关闭 TCP 通信端口
}
void send_initial();        //发送初始化函数
void client_ready();        //客户端初始化函数
void send(float *data,const int &datanum);          //发送浮点型数组函数
void send(const float &data);                       //发送单个浮点数函数
void send(const double &data);                      //发送单个双精度数函数
```

4.2 tcp_receive 类

1）类内属性：类内属性包括公有属性和私有属性。其中，公有属性有 TCP 通信的 socket 号和通信初始化时的发送接收缓存区，私有属性有通信地址。类属性具体命名如下：

```
int fd;                 //client 端的 socket 号
int socket_fd;          //server 端的 socket 号
char buffer[255];           //接收缓存区
double hot;                 //发送缓存区
struct sockaddr_in addr;            //通信地址
```

2）类内函数：类内函数包括构造函数、析构函数、接收初始化函数、服务端初始化函数、接收浮点型数组函数、接收单个浮点数函数和接收单个双精度数函数。其中，接收初始化函数完成与另一台计算机的通信连接，构造函数调用了接收初始化函数，完成接收端的初始化工作，析构函数执行关闭 TCP 通信端口的操作。类函数具体如下：

```
tcp_receive()               //构造函数
{
    receive_initial();              //调用类内接收初始化函数
```

```
}
~tcp_receive()              //析构函数
{
    close(socket_fd);               //关闭 TCP 通信端口
}
void receive_initial();         //接收初始化函数
void server_ready();            //服务端初始化函数
void receive(float *data,const int &datanum);          //接收浮点型数组函数
void receive(float &data);                             //接收单个浮点数函数
void receive(double &data);                            //接收单个双精度数函数
```

5 性能测试

5.1 演示/测试环境的搭建

1）分别在服务端和客户端的计算上安装 cuda，在本次实验中，服务端计算机安装的 cuda 版本为 cuda 11.5，客户端计算机安装的 cuda 版本为 cuda 11.0。

2）将两台计算机用网线连接，并设置为局域网通信模式。服务端在网络设置中将 IPv4 设置为手动模式，并将其地址设置为 192.168.1.106，子网掩码设置为 255.255.255.0，网关设置为 192.168.1.10。客户端在网络设置中将 IPv4 设置为手动模式，并将其地址设置为 192.168.1.112，子网掩码设置为 255.255.255.0，网关设置为 192.168.1.10。

5.2 操作步骤

1）分别在两台计算机上的单机文件夹 SingComputer 下打开终端，为了防止栈溢出，首先通过运行 ulimit –s 1048576 指令来开辟足够的栈空间，接着通过 g++ main.cpp –o main 指令编译 main.cpp 文件，编译完成后在终端中通过./main 指令运行单机程序 5 次，记下每一次单机计算所需的时间。

2）在服务端的 DistributedComputing_Server 文件夹下打开终端，为了防止栈溢出，首先通过运行 ulimit –s 1048576 指令来开辟足够的栈空间，接着通过 nvcc –lcublas main.cu –o main 指令编译 main.cu 文件，编译完成后在终端通过./main 指令运行服务端程序，等待客户端的连接。

3）在客户端的 DistributedComputing_Client 文件夹下打开终端，为了防止栈溢出，首先通过运行 ulimit –s 1048576 指令来开辟足够的栈空间，接着通过 nvcc –lcublas main.cu –o main 指令编译 main.cu 文件，编译完成后在终端通过./main 指令运行客户端程序。至此服务端和客户端连接上开始计算数据。

4）按步骤 2 和步骤 3 运行五次，记录每一次并行计算所需的时间。

5.3 测试结果及评价

按照上述步骤运行程序。单机运行时，第一台计算机在终端运行时的结果如图 5.1 所示，第二台计算机在终端运行时的结果如图 5.2 所示。多机运行时，服务端的运行结果如图 5.3 所示。

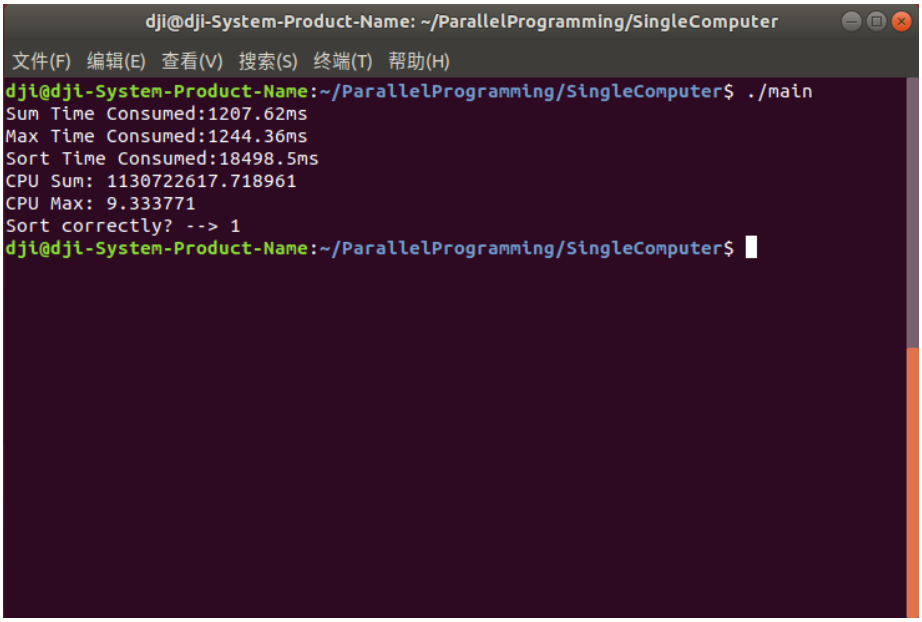

图 5.1 单机运行时第一台计算机的运行结果

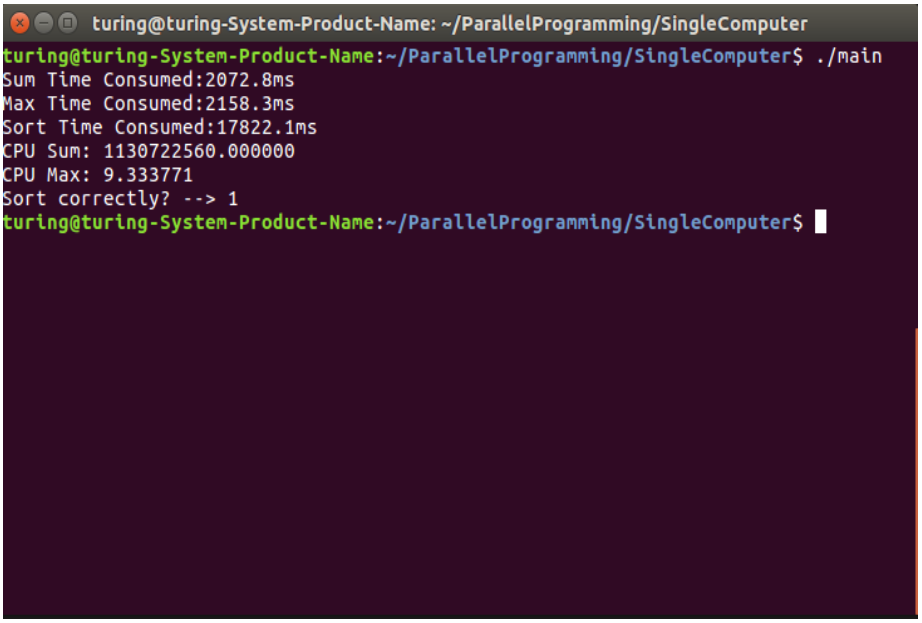

图 5.2 单机运行时第二台计算机的运行结果

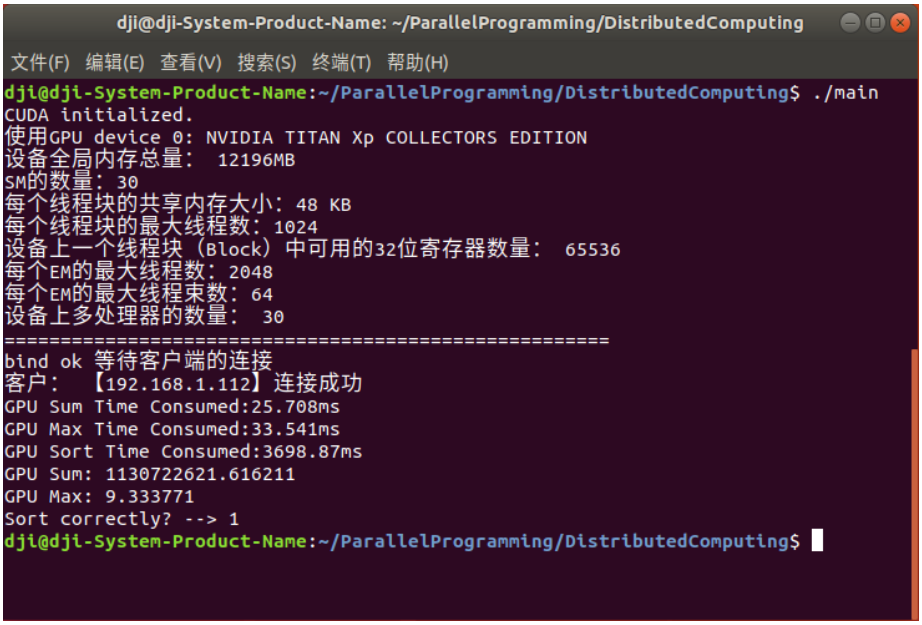

图 5.3 多机运行时服务端的运行结果

单机运行时，服务端的计算耗时见表 1，客户端的计算耗时见表 2。两台计算机并发执行时，计算耗时见表 3。单机运行与并行计算的耗时对比见表 4。由实验结果可见，对于求和与求最大值，使用两台电脑并行计算并且采用 cuda 加速的方式，其计算速度相比单机单线程大大提高，加速比为几十倍。对于排序算法，在 64*2000000 的数据量下，若采用冒泡排序等简单的排序算法，单机运行的时间很长，无法在有限的时间内完成排序，其加速比固然很大，本实验单机和并行计算都采用了归并排序算法，归并排序本身具有加速排序的效果，使用两台电脑并行计算并且采用 cuda 加速的方式，相比单机单线程的计算方式，其加速比约为 5 倍。总体加速效果较好。

表 1、服务端单机运行耗时表

次数	求和	求最大值	排序
1	1207.62ms	1244.36ms	18498.5ms
2	1214.46ms	1268.66ms	19107.1ms
3	1229.64ms	1264.47ms	18618.8ms
4	1208.48ms	1243.56ms	18592ms
5	1209.85ms	1246.51ms	18959.8ms
平均	1214.01ms	1253.51ms	18755.24ms

表 2、客户端单机运行耗时表

次数	求和	求最大值	排序
1	2072.8ms	2158.3ms	17822.1ms
2	2070.61ms	2206.89ms	18690.8ms
3	2085.98ms	2179.94ms	18083.1ms
4	2078.5ms	2170.47ms	17820.5ms
5	2071.7ms	2162.77ms	17877ms
平均	2075.92ms	2175.67ms	18058.7ms

表 3、并行计算耗时表

次数	求和	求最大值	排序
1	25.708ms	33.541ms	3698.87ms
2	25.792ms	34.057ms	3712.19ms
3	26.953ms	33.478ms	3696.97ms
4	25.89ms	33.76ms	3698.07ms
5	26.974ms	39.226ms	3698.38ms
平均	26.263ms	34.812ms	3700.90ms

表 4、单机运行与并行计算耗时对比表

函数	计算方式	服务端	客户端
求和	单机	1214.01ms	2075.92ms
	并行	26.263ms	
	加速比	46.23	79.04
求最大值	单机	1253.51ms	2175.67ms
	并行	34.812ms	
	加速比	36.01	62.50
排序	单机	18755.24ms	18058.7ms
	并行	3700.90ms	
	加速比	5.07	4.88

6 总结和分析

本次作业通过两台计算机以及多线程的方式对求和、求最大值和排序函数进行加速，为了达到更大的加速比，我们小组采用了 cuda 进行加速。在程序的设计过程中，求和与求最大值采用了并行归约算法，排序函数采用归并排序算法。此外，我们将数据分布在两台计算机上进行并行计算，通过 TCP 通信将其中一台计算机的计算结果发送到另外一台计算机上，较大地提高了计算的速度，加速效果较好。不足之处在于将数据总量为 64*1000000 的排序序列从一台计算机传输到另一台计算机上的耗时较大，若能对传输速度进一步优化，排序函数的加速比能进一步提高。通过此次作业，我们小组将课堂上学到的知识和技能较好地应用到了实际的问题中，对并行编程，多机通信，cuda 编程等知识有了更好的认识并得到了较好的巩固，受益匪浅。

7 程序使用说明

（1）单机运行程序在 SingleComputer 文件夹下，采用 g++ main.cpp –o main 指令编译生成可执行文件 main。

（2）多机运行，服务端程序在 DistributedComputing_Server 文件夹下，客户端程序在 DistributedComputing_Client 文件夹下。多机运行时，首先需要将服务端代码中的 tcp.h 文件中（第 124 行）的 receive_initial()函数的 addr.sin_addr.s_addr = inet_addr("192.168.1.106")代码中的 IP 地址改为本机的 IP 地址，还需将客户端代码中的 tcp.h 文件中（第 84 行）的 send_initial()函数中的 addr.sin_addr.s_addr = inet_addr("192.168.1.106")代码中的 IP 地址改为

服务端的 IP 地址，并且需要保证服务端和客户端的端口号一致。

（3）多机运行时服务端和客户端的程序使用 nvcc –lcublas main.cu –o main 指令进行编译，使用./main 执行。

（4）为了防止栈溢出，在运行程序前可执行 ulimit –s 1048576 指令来开辟足够的栈空间。

8 人员分工

序号	工作内容	参与成员
1	单机程序开发	黄定梁
2	双机求和加速模块开发	黄定梁、陈俊柱
3	双机求最大值加速模块开发	
4	双机排序加速模块开发	
5	TCP 协议双机通讯模块开发	陈俊柱
6	代码调试和封装	黄定梁、陈俊柱
7	报告撰写	

9 参考文献

[1] C++八大排序算法, https://blog.csdn.net/qq_33596574/article/details/88827925

[2] Linux vscode 开大栈空间, https://www.cnblogs.com/GK0328/p/13673855.html

[3] CUDA C++ Programming Guide, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[4] CUDA C++ Best Practices Guide, https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

[5] CUDA 编程第三章：CUDA 执行模型，https://blog.csdn.net/qq_42683011/article/details/113593860?spm=1001.2014.3001.5501

[6] CUDA 编程第五章：共享内存 & 常量内存，https://blog.csdn.net/qq_42683011/article/details/113820683?spm=1001.2014.3001.5501

10 附件和提交物检查清单（Check List）

序号	说明	是否已提交
1	课程设计报告	是
2	单机程序源代码 SingleComputer（在 Linux 下的 VSCode 打开）	是
3	双机程序源代码 DistributedComputing_Client 和 DistributedComputing_Server（在 Linux 下的 VSCode 打开）	是