

# 实验环境配置

李奕琛

February 2020

## 目录

<b>1</b>	<b>SIMD, 多线程的开发环境</b>	<b>3</b>
1.1	编译器 MinGW-W64 . . . . .	3
1.2	集成开发环境:Code::Blocks . . . . .	3
1.2.1	编译器选择 . . . . .	4
1.2.2	创建项目 . . . . .	5
1.2.3	编译选项设置 . . . . .	6
1.2.4	编译/运行项目 . . . . .	8
<b>2</b>	<b>性能测试方法</b>	<b>9</b>
2.1	问题规模 . . . . .	9
2.2	测试数据 . . . . .	9
2.3	样例 . . . . .	9
2.4	Windows 下精确计时 . . . . .	13
<b>3</b>	<b>MPI 开发环境</b>	<b>15</b>
3.1	示例 . . . . .	16
3.1.1	作业运行提交 . . . . .	16
3.1.2	程序名.pbs.sh 文件构成 . . . . .	17
<b>4</b>	<b>GPU 上机实验说明</b>	<b>18</b>
4.1	课程兑换 . . . . .	18
4.2	课程加入 . . . . .	20

## 1 SIMD, 多线程的开发环境

硬件: 同学们自己的笔记本即可

软件: Visual C++ 或者 MinGW+Code Blocks。接下来我们介绍 MinGW+Code Blocks 的配置

### 1.1 编译器 MinGW-W64

下载地址: <http://www.mingw-w64.org/doku.php>

MinGW 全称 Mininmailist GUN for Windows。是极简的 Windows 平台原生应用开发套件。实际上是将经典的开源 C 语言编译器 GCC 移植到了 Windows 平台下。MinGW-w64 是原始 MinGW 升级版, 可以编译生成 64 位或者 32 位的可执行文件, 直接支持 SIMD, 多线程 (安装时选择 posix 线程) 等并行编程

### 1.2 集成开发环境:Code::Blocks

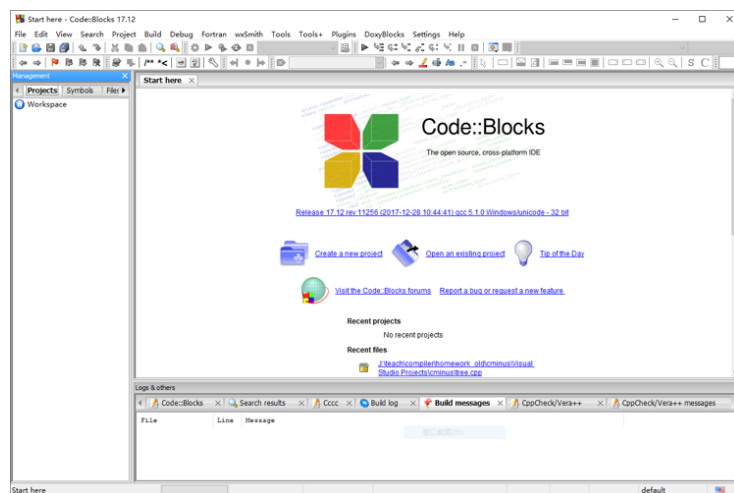


图 1.1: Code::Blocks 主界面

下载地址: <http://www.codeblocks.org/downloads/>

Code::Blocks 是免费开源的跨平台集成开发环境,支持多种编译器:GCC,MSVC++,clang 等。具有强大的调试功能,完整的 GDB 指出以及 MS CDB 的部分特性。

### 1.2.1 编译器选择

安装好的 Code::Blocks 需要设置编译器,如下图所示:

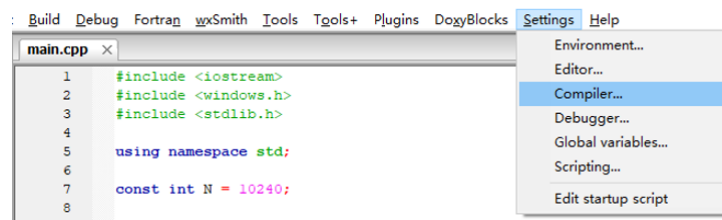


图 1.2:

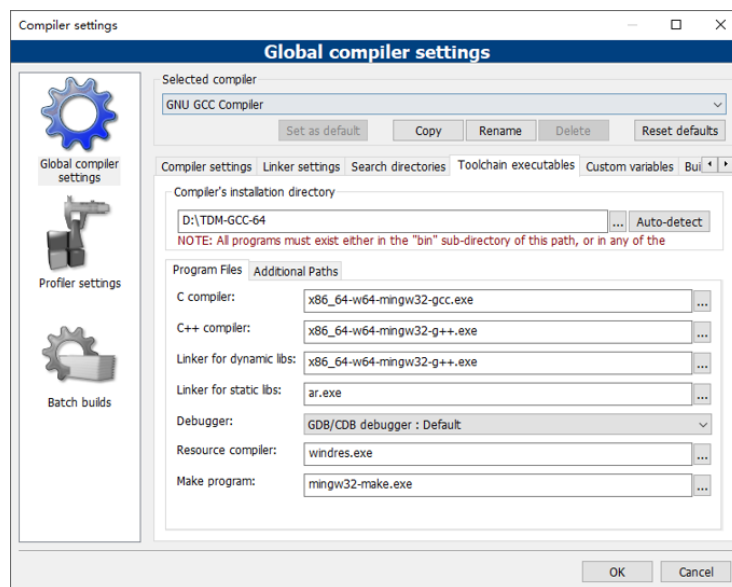


图 1.3:

### 1.2.2 创建项目

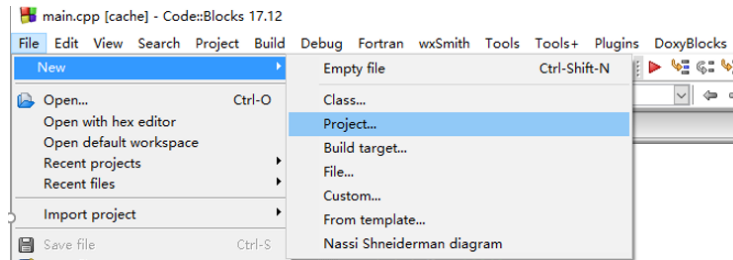


图 1.4: 创建项目

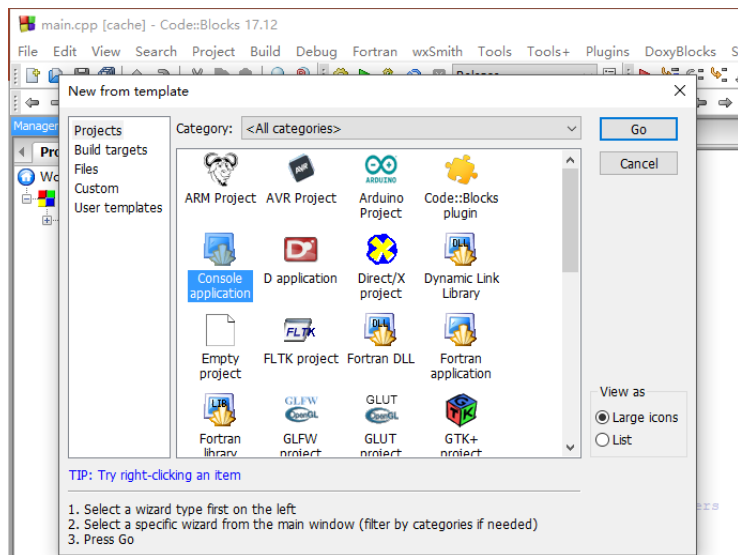


图 1.5: 创建项目

### 1.2.3 编译选项设置

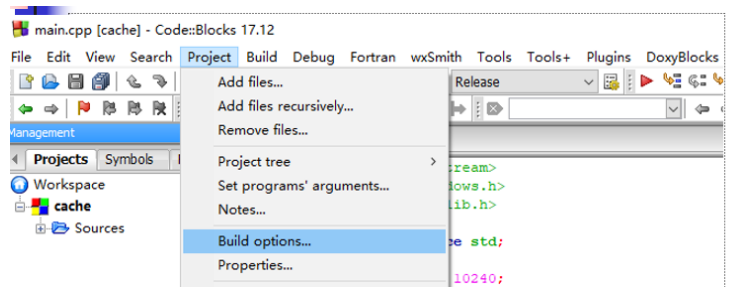


图 1.6: 编译选项

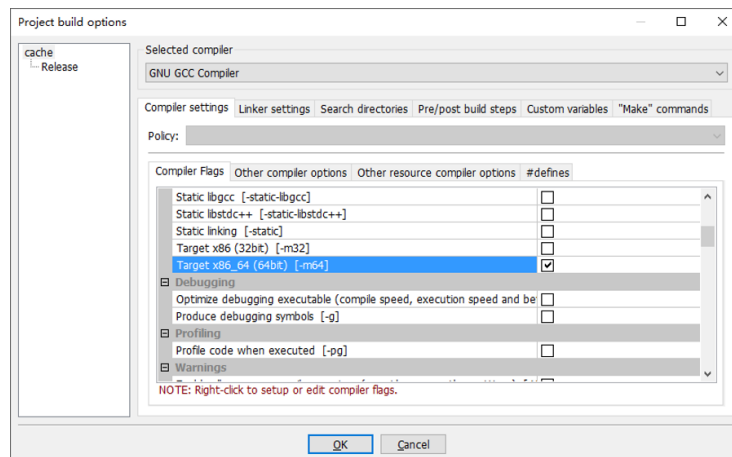


图 1.7: 32 位/64 位设置

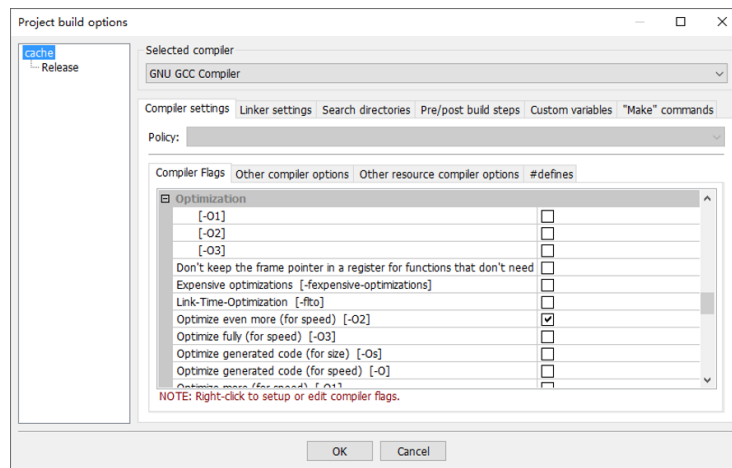


图 1.8: 优化力度设置

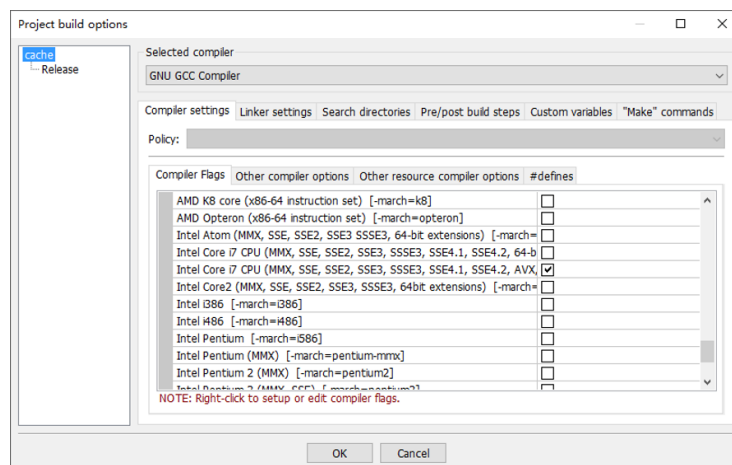


图 1.9: 系统架构设置

### 1.2.4 编译/运行项目

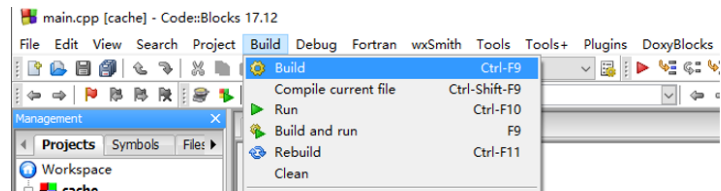


图 1.10: 编译项目

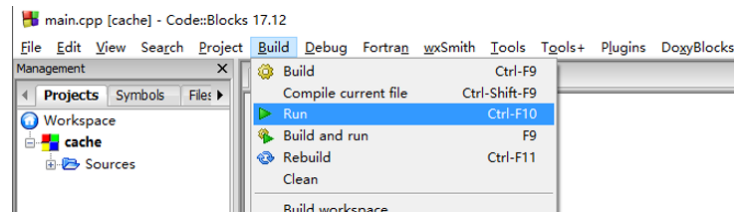


图 1.11: 运行项目

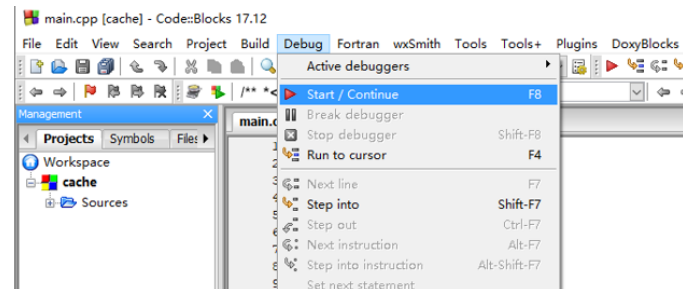


图 1.12: 调试项目



## 2 性能测试方法

测试程序的性能，我们需要得到程序实际需要的空间和时间，其中编译锁需空间时间可以不考虑。

计时的机制有：

跨平台：clock() / CLOCKS\_PER\_SEC，精度不高

Linux：gettimeofday()

Windows：QueryPerformance

测试方法：(1) 给定问题规模  $n$ ；(2) 按需求设计测试数据

### 2.1 问题规模

问题规模的确定因素为执行时间和执行次数，即对于一个程序，我们究竟需要测试多少次，每次测试多少数据。以插入排序为例子，其中最坏的情况为  $\Theta(n^2)$ ，平方函数至少需要 3 个点，并且在测试时需要 3 个以上的  $n$ ， $n$  需要渐进的取到，同时当  $n$  取值较小时，同一个  $n$  需要测试多次来取到精确的结果。一种可能的规模设计为：

$n=0-100$  精细的测试：0,10,20 ... 100

$n>100$  稀疏：200,300,400...

### 2.2 测试数据

测试数据的设计需要仔细思考，如果测试数据过于特别，将无法得到程序真正的性能。以输入排序为例，最坏以及最好的情况下测试数据分别为递减序列和递增序列。但是如果我们需要测试程序的平均情况，这时候需要随机产生大量的测试数据。

### 2.3 样例

以插入排序为例子，如果我们需要测试插入排序的最坏情况，测试数据就需要是递减序列

```
void main(void)
{
```

```
int a[1000], step = 10;
clock_t start, finish;
for (int n = 0; n <= 1000; n += step) {
    // get time for size n
    for (int i = 0; i < n; i++)
        a[i] = n - i; // initialize
    start = clock();
    InsertionSort(a, n);
    finish = clock();
    cout << n << ' '
    << (finish - start)/float(CLOCKS_PER_SEC) << endl;
    if (n == 100) step = 100;
}
```

测试规模我们取 n=10,20...100,200...1000, 测试结果如下图所示

n	时间 (s)	n	时间 (s)
0	0	100	0.06
10	0	200	0
20	0	300	0
30	0	400	0
40	0	500	0
50	0	600	0.05
60	0	700	0.06
70	0	800	0.05
80	0	900	0.06
90	0	1000	0.11

图 2.13: 测试结果

同时我们需要重复测试，从而提高精度。程序可以改为：

```
void main(void)
{
    int a[1000], n, i, step = 10;
    long counter;
    float seconds;
    clock_t start, finish;
    for (n = 0; n <= 1000; n += step) {
        // get time for size n
        start = clock(); counter = 0;
        while (clock() - start < 10) {
            counter++;
            for (i = 0; i < n; i++)
                a[i] = n - i; // initialize
            InsertionSort(a, n);
        }
    }
}
```

```
    }  
    finish = clock();  
    seconds = (finish - start)/float(CLOCKS_PER_SEC);  
    cout << n << ' ' << counter << ' ' << seconds  
          << ' ' << seconds / counter << endl;  
    if (n == 100) step = 100;}  
}
```

这样得到的测试结果中，当  $n$  取值较小时，需要重复多次来获得更加精确的结果，测试结果如下图所示：

n	重复次数	总时间 (s)	每次排序时间 (s)
0	65	0.06	0.000923077
10	294	0.05	0.000170068
20	348	0.06	0.000172414
30	145	0.05	0.000344828
40	120	0.06	0.0005
50	179	0.05	0.00027933
60	141	0.06	0.000425532
70	70	0.05	0.000714286
80	99	0.06	0.000606061
90	60	0.05	0.000833333
100	60	0.06	0.001
200	12	0.05	0.00416667
300	9	0.06	0.00666667
400	4	0.05	0.0125
500	3	0.06	0.02
600	2	0.05	0.025
700	1	0.06	0.06
800	1	0.05	0.05
900	1	0.11	0.11

图 2.14: 测试结果

## 2.4 Windows 下精确计时

windows 下精确计时可以调用 windows.h 库中的 QueryPerformance 相关函数进行计时，具体如下：

```
#include <iostream>
#include <windows.h>
#include <stdlib.h>

using namespace std;

const int N = 10240;           // matrix size

Double b[N][N], col_sum[N];

Void init(int n)               // generate a N*N matrix
{
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            b[i][j] = i + j;
}
```

```
int main()
{
    long long head, tail, freq;          // timers
    init(N);
    // similar to CLOCKS_PER_SEC
    QueryPerformanceFrequency((LARGE_INTEGER *)&freq);
    // start time
    QueryPerformanceCounter((LARGE_INTEGER *)&head);
    for (int i = 0; i < 1000; i++) {
        col_sum[i] = 0.0;
        for (int j = 0; j < 1000; j++)
            col_sum[i] += b[j][i];
    }
    // end time
    QueryPerformanceCounter((LARGE_INTEGER *)&tail);
    cout << "Col: " << (tail - head) * 1000.0 / freq
    << "ms" << endl;
}
```

### 3 MPI 开发环境

课程 MPI 编程实验将在基于金山云的虚拟机集群上运行，其架构如下图所示：

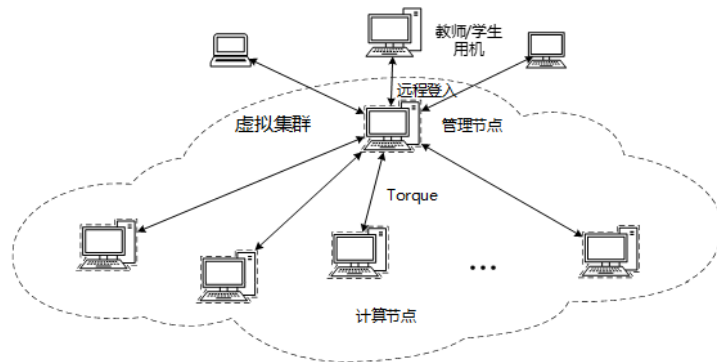


图 3.15: 虚拟机集群

将在讲授 MPI 编程时开通虚拟机集群，采用管理软件 Torque，这是基于 pbs 的一个开源版本的分布式作业提交系统。开通虚拟机集群后会发给大家 IP 地址以及用户名密码。登录管理节点，将程序编译成功后得到可执行文件。输入命令：

```
pbsmpirun 程序路径 线程数
```

即可自动生成 pbs 脚本：程序名.pbs.sh 并提交作业。提交后会得到一个作业编号。输入 qstat 可以看到目前队列中作业的状态：

Q-正在排队 R-正在运行 C-运行完毕。

当运行完成后，目录下会多两个文件：执行文件名.o 作业编号，执行文件名.e 作业编号，分别代表标准输出和错误输出。

### 3.1 示例

#### 3.1.1 作业运行提交

以 MPI 自带的 cpi 程序为例，输入命令

```
pbsmpirun cpi 4
qstat
qstat
```

得到如下结果：

```
[pbs@master test]$ pbsmpirun cpi 4
Submit job (excute file = cpi, number of process = 4).
129.master
[pbs@master test]$ qstat
Job ID          Name          User          Time Use S Queue
-----
127.master      cpi           pbs           00:00:00 C batch
128.master      cpi           pbs           00:00:00 C batch
129.master      cpi           pbs           0 R batch
[pbs@master test]$ qstat
Job ID          Name          User          Time Use S Queue
-----
127.master      cpi           pbs           00:00:00 C batch
128.master      cpi           pbs           00:00:00 C batch
129.master      cpi           pbs           00:00:00 C batch
[pbs@master test]$ ls
cpi cpi.e129 cpi.o129 cpi.pbs.sh
```

图 3.16: 提交作业示例

可以看到作业编号为 129。开始状态为 R，表示正在运行。之后状态为 C，表示运行结束。而 128.master 是之前提交的作业。这时当前目录下会多出两个文件：cpi.o129 cpi.e129。cpi.pbs.sh 是 pbsmpirun 命令自动生成的脚本。打开 cpi.o129 可以看到程序的标准输出：

```
Process 0 of 4 is on node4
Process 2 of 4 is on node2
Process 1 of 4 is on node3
pi is approximately 3.1415926544231239, Error is 0.0000000008333307
wall clock time = 0.001947
Process 3 of 4 is on node1
```

图 3.17: cpi.o129



打开 `cpi.e129` 可以看到程序的错误输出：

```
[1] 21:17:59 [SUCCESS] node4
[2] 21:17:59 [SUCCESS] node3
[3] 21:17:59 [SUCCESS] node2
[4] 21:17:59 [SUCCESS] node1
[1] 21:18:00 [SUCCESS] node4
[2] 21:18:00 [SUCCESS] node2
[3] 21:18:00 [SUCCESS] node1
[4] 21:18:00 [SUCCESS] node3
```

图 3.18: `cpi.e129`

### 3.1.2 程序名.pbs.sh 文件构成

以 `cpi` 生成的 `cpi.pbs.sh` 文件为例：`cpi.pbs.sh` 会被发送到某一个计算

```
#PBS -N cpi
#PBS -l nodes=4
pssh -h $PBS_NODEFILE mkdir -p /home/pbs/test 1>&2
scp master:/home/pbs/test/cpi /home/pbs/test
pscp.pssh -h $PBS_NODEFILE /home/pbs/test/cpi /home/pbs/test 1>&2
/usr/local/bin/mpirun -np 4 -machinefile $PBS_NODEFILE /home/pbs/test/cpi
```

图 3.19: `cpi.pbs.sh`

节点上运行，效果等效于 `sh cpi.pbs.sh`。文件含义如下：

```
#PBS -N cpi 任务名称为 cpi
#PBS -l nodes=4 需要四个节点来计算。
pssh -h $PBS_NODEFILE mkdir -p /home/pbs/test 1>&2
在分配到的 4 个计算节点上创建对应的路径。
scp master:/home/pbs/test/cpi /home/pbs/test
pscp.pssh -h $PBS_NODEFILE /home/pbs/test/cpi /home/pbs/test 1>&2
获取 master 节点中的 cpi 程序，并分发到每个计算节点。

/usr/local/bin/mpirun -np 4 -machinefile $PBS_NODEFILE /home/pbs/test/cpi
在 4 个计算节点上运行 mpi 程序 cpi。
```

## 4 GPU 上机实验说明

### 4.1 课程兑换

进入网站：<https://courses.nvidia.com/courses/course-v1:DLI+C-AC-01+V1-ZH/about>

打开页面点击 Enroll Now, 需要先注册账号。创建完账户后, 在“购物车和付款信息”页点击“输入促销码”, 应用一下优惠码, 可以免去费用。优惠码是:

LITEACH1218\_60\_Sx

注意: 此处虽然不交钱, 但账单地址还是要填, 并不真正邮寄。如下图所示:

The screenshot shows the '购物车与付款信息' (Cart and Payment Information) page. At the top, it says '你的购物车' (Your Cart) with an order number '14136153510' and a total of '619.34CNY'. Below this, there's a table with one item: 'Fundamentals of Accelerated Computing with CUDA C/C++'. A red circle highlights the '输入促销码' (Enter promo code) link. Below the cart, there's a section for '账单地址' (Billing Address) with a dropdown for '地址簿' (Address Book). The '顾客类型' (Customer Type) is set to '个人购买' (Personal Purchase). The form includes fields for '名字' (Name), '姓氏' (Last Name), '公司名称' (Company Name), '电话号码' (Phone Number), '地址1' (Address 1), '地址2' (Address 2), '城市' (City), '州/省' (State/Province), '国家' (Country), and '邮政编码' (Zip Code). There are also dropdowns for '州/省' and '国家'.

图 4.20:

### 购物车与付款信息

[继续购物](#)

#### 你的购物车



Fundamentals of Accelerated Computing with CUDA C/C++

1

删除

订单: 14136153510

619.34CNY 0.00CNY

小计: 0.00CNY

促销代码:  [应用](#)

#### 账单地址

顾客类型: ☒ 个人购买 ☐ 企业购买

地址簿: 

▼

名字: \*

地址1: \*

地址2:

地址3:

城市: \*

州/省: \*

邮政编码: \*

国家: \*


电话号码: \*

选择一个

图 4.21:

### 验证订单


#### 账单地址



编辑

你的购物车

订单: 14136153510



Fundamentals of Accelerated Computing with CUDA C/C++

数量: 1

619.34CNY 0.00CNY

编辑

小计: 0.00CNY

税金: 0.00CNY

总计: 0.00CNY

[提交](#)

图 4.22:

## 4.2 课程加入

在“收据”页，点击“proceed to courses”，进入课程页。



图 4.23:



图 4.24:



图 4.25:

“课程”标签里下，会发现“加速计算基础—CUDA C/C++”课程一共有 3 个小节。需要依次完成每个小节。点击各小节标题可进入相应小节。每个小节进入后，有详细的说明，请大家仔细阅读。有一个类似“播放”的“START”按钮，点击后可加载相应的实验环境，需要等 5-10 分钟左右时间才能加载成功，成功后会显示“2 小时倒计时”和“Launch Task”按钮，点击“Launch Task”按钮，开始本课程的学习。依次学习完 3 个小节后，最后有一个作业任务，完成后可获得相应的学习证书。

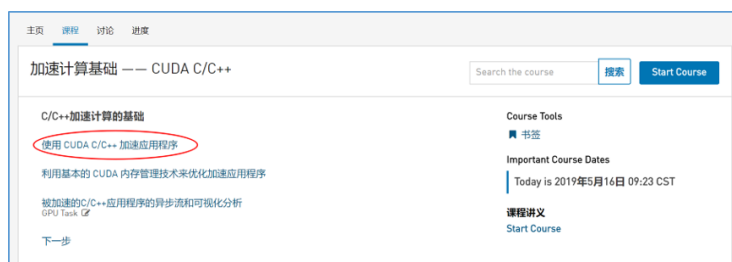


图 4.26:



图 4.27:



图 4.28:

点击“Launch Task”，在新开的窗口中进行学习，请仔细阅读文中的内容，里边有程序如何编译、执行的说明：



图 4.29: