

---

# 异构计算课程 结 题 报 告

报告名称: 基于 CUDA 的 GPU 计算 PI 值

姓 名: 郭思齐

学 号: 3019244140

联系电话: 13021304295

电子邮箱: 1811400940@qq.com

填写日期: 2021-9-26

2021 年制

---

## 基于 CUDA 的 GPU 计算 PI 值

### 一、实验内容概述

该实验要求编写 CUDA 的程序以使用 GPU 协处理器实现 PI 值的计算，通过 Xftp7 上传文件到平台，使用 Xshell7 登陆学校平台运行 CUDA 程序并进行 CPU 和 GPU 运行性能、运行时间的比较，总结得到实验结论。

异构计算环境，远程登陆学校平台 GPU 服务器--会话 172.26.17.240: 22；使用并依次输出 CPU, GPU 程序的编译运行后 PI 的值以及程序运行时间。

在最后的的分析方面需要比较两种并行算法的 CPU 和 GPU 环境下计算 PI 值的精确度、运行所消耗的时间以及加速比。凭借这些参数来探究 GPU 对计算的性能提升。

### 实验相关——CUDA:

2006 年，NVIDIA 公司发布了 CUDA (Compute Unified Device Architecture)，其是一种新的操作 GPU 计算的硬件和软件架构，是建立在 NVIDIA 的 GPU 上的一个通用并行计算平台和编程模型，它提供了 GPU 编程的简易接口，基于 CUDA 编程可以构建基于 GPU 计算的应用程序，利用 GPU 的并行计算引擎来更加高效地解决比较复杂的计算难题。它将 GPU 视作一个数据并行计算设备，而且无需把这些计算映射到图形 API。操作系统的多任务机制可以同时管理 CUDA 访问 GPU 和图形程序的运行库，其计算特性支持利用 CUDA 直观地编写 GPU 核心程序。

CUDA 提供了对其它编程语言的支持，如 C/C++，Python 等语言。只有安装 CUDA 才能够进行复杂的并行计算。主流的深度学习框架也都是基于 CUDA 进行 GPU 并行加速的，几乎无一例外。

## 二、算法分析设计

方法一——积分法：（矩阵法则的数值积分方法估算 PI 的值）

$$\text{计算公式: } \pi = \int_0^1 \frac{4}{1+x^2} dx \approx \sum_{0 \leq i \leq N} \frac{4}{1+(\frac{i+0.5}{N})^2} \times \frac{1}{N}$$

方法二——幂级法：通过数学方法幂级展开

$$\text{计算公式: } \pi = 4 \times \arctan(1) = 4 \times (1 - \frac{1}{3} + \frac{1}{5} - \dots + \frac{(-1)^{n+1}}{2n-1} - \dots)$$

## 三、实验数据分析

### 3.1 实验环境

```
$ cat /proc/cpuinfo | grep "physical id" | uniq | wc -l
```

```
1
```

```
$ cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq -c
```

```
12 Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
```

```
$ cat /proc/meminfo | grep MemTotal
```

```
MemTotal: 32846900 kB
```

```
$ uname -a # 查看内核/操作系统/CPU 信息的 linux 系统信息命令
```

```
Linux hpcgpu 3.10.0-1062.4.1.el7.x86_64 #1 SMP Fri Oct 18 17:15:30 UTC 2019 x86_64  
x86_64 x86_64 GNU/Linux
```

```
$ nvidia-smi # 查看显存使用情况
```

```
Mon Sep 27 19:18:36 2021
```

NVIDIA-SMI 418.67				Driver Version: 418.67		CUDA Version: 10.1	
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	
0	Tesla K40c	Off	00000000:04:00.0	Off	0		
27%	55C	P0	63W / 235W	80MiB / 11441MiB	0%	Default	
Processes:				GPU Memory			
GPU	PID	Type	Process name	Usage			
0	12364	C	./cuda_pi_2	69MiB			

```
$ watch -n 10 nvidia-smi
```

```
Mon Sep 27 19:21:49 2021
+-----+
| NVIDIA-SMI 418.67      Driver Version: 418.67      CUDA Version: 10.1      |
+-----+-----+-----+-----+-----+-----+
| GPU  Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+-----+
|   0   Tesla K40c          Off          | 00000000:04:00:0 | Off           0      |
| 42%    72C    P0      152W / 235W | 147MiB / 11441MiB | 100%      Default   |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| Processes:                                     GPU Memory |
|  GPU       PID    Type    Process name                     Usage      |
+-----+-----+-----+-----+-----+-----+
|    0       12364    C      ./cuda_pi_2                          69MiB      |
|    0       12576    C      ./cuda_pi_2                          64MiB      |
+-----+-----+-----+-----+-----+-----+
```

3.2 实验数据综合分析

积分法——实验结果数据统计:

分块数 N	1024*1024*64 (CPU 串行)
Threads per Blocks	0
Blocks per Grid	0
PI 的值	3.14159265359101436133
运行时间	584.815979ms

Threads per Blocks 为自变量

1024*1024*64	1024*1024*64	1024*1024*64	1024*1024*64
256	128	64	32
64	64	64	64
3.14159265358979222782	3.14159265358979311600	3.14159265358979444827	3.14159265358979000737
2.189633427ms	3.094176ms	5.621504ms	10.663008ms

Blocks per Grid 为自变量

1024*1024*64	1024*1024*64	1024*1024*64	1024*1024*64
256	256	256	256
32	128	256	512
3.14159265358979444827	3.14159265358979267191	3.14159265358979356009	3.14159265358979356009
3.101440ms	2.380448ms	2.138688ms	2.102144ms

N 为自变量

分块数 N	1024*1024*128 (CPU 串行)	1024*1024*256 (CPU 串行)
Threads per Blocks	0	0
Blocks per Grid	0	0
PI 的值	3.14159265358980555050	3.14159265358929218337
运行时间	1131.520996ms	2223.868896ms

N 为自变量时 GPU 的性能表现:

1024*1024*128	1024*1024*256	1024*1024*512	1024*1024*1024
256	256	256	256
64	64	64	64
3.14159265358979311600	3.14159265358979444827	3.14159265358979133964	3.14159265358979000737
4.053536ms	7.772352ms	14.862432ms	29.343744ms

幂级法——实验结果数据统计:

分块数 N	1024*1024*64 (CPU 串行)
Threads per Blocks	0
Blocks per Grid	0
PI 的值	3.14159263868885840765
运行时间	602.540039ms

Threads per Blocks 为自变量

1024*1024*64	1024*1024*64	1024*1024*64	1024*1024*64
256	128	64	32
64	64	64	64
3.14159263868862659308	3.14159263868863147806	3.14159263868866078795	3.14159263868859506275
2.467168ms	3.243040ms	5.640864ms	10.851232ms

Blocks per Grid 为自变量

1024*1024*64	1024*1024*64	1024*1024*64	1024*1024*64
256	256	256	256
32	128	256	512
3.14159263868863281033	3.14159263868862925762	3.14159263868863058988	3.14159263868863547486
3.480640ms	2.563072ms	2.254496ms	2.253664ms

N 为自变量

分块数 N	1024*1024*128 (CPU 串行)	1024*1024*256 (CPU 串行)
Threads per Blocks	0	0
Blocks per Grid	0	0
PI 的值	3.14159264613847843961	3.14159264986401165487
运行时间	1133.812988ms	2234.450928ms

1024*1024*128	1024*1024*256	1024*1024*512	1024*1024*1024
256	256	256	256
64	64	64	64
3.14159264613920052867	3.14159264986448594215	3.14159265172715462811	3.14159265265844345194
4.470304ms	8.471744ms	16.576481ms	33.286625ms

由表中数据可以看出基于 CUDA 的 GPU 相比 CPU 计算性能得到极大的提升，可以达到十几倍甚至几十倍。

随着 Threads per Blocks 和 Blocks per Grid 的增大，计算所消耗的时间在减小，但是减小的幅度越来越不明显。

但是在划分 N 不变的情况下，计算的迭代次数不变，因此 CPU 的计算时间相差不大。而 N 的大小大致与 CPU 或者 GPU 的计算时间正相关。

积分法和幂级展开法两种算法相比较，计算所消耗的时间大致相当，只是整体而言，针对每一次控制变量的实验来说，积分法会比幂级展开法消耗的时间略小一些。

### 加速比的分析:

在并行计算课程中，因为仅仅是多线程的比较，我们使用

$$S_p = \frac{T_1}{T_p}$$

来计算加速比，但是在本次实践中，加速比计算会有一些不同。

使用 GPU 后的加速比的比较会有很多种版本，在这里，我们只用统一的服务器进行运算，编译条件以及运行时条件一致，因此不考虑计算平台的影响。

有一种加速比计算公式为：

$$Sp = CPU\_run\_time / (CPU\_run\_time + GPU\_run\_time)$$

积分法:

(1) 分块数: 1024\*1024\*64    Blocks per Grid: 64

Threads per Block	0	32	64	128	256
运行时间	584.815979ms	10.663008ms	5.621504ms	3.094176ms	2.189633427ms
加速比	1	0.982093	0.990479	0.994737	0.99627

(2) 分块数: 1024\*1024\*64    Threads per Block: 256

Blocks per Grid	0	32	64	128	256	512
运行时间	584.815979ms	3.101440ms	2.189633427ms	2.380448ms	2.138688ms	2.102144ms
加速比	1	0.994725	0.99627	0.995946	0.996356	0.996418

---

(3) N 作为分的块数，在计算中作为迭代次数存在，不再计算加速比

幂级展开法：

(1) 分块数：1024\*1024\*64    Blocks per Grid: 64

Threads per Block	0	32	64	128	256
运行时间	602.540039ms	10.851232ms	5.640864ms	3.243040ms	2.467168ms
加速比	1	0.981783	0.990447	0.994485	0.995799

(2) 分块数：1024\*1024\*64    Threads per Block: 256

Blocks per Grid	0	32	64	128	256	512
运行时间	602.540039ms	3.480640ms	2.467168ms	2.563072ms	2.254496ms	2.253664ms
加速比	1	0.994084	0.995799	0.995636	0.99616	0.996161

(3) N 作为分的块数，在计算中作为迭代次数存在，不再计算加速比

### 加速比数据分析：

GPU 的加速比整体趋势是：随单位 grid 下的 Blocks 的数量增长和  
单位 block 数量下 threads 的增长而增加。比较奇怪的是在 Threads per  
Block: 256 的时候，Blocks per Grid 为 64 的条件下，加速比达到了一个  
局部极大值，



---

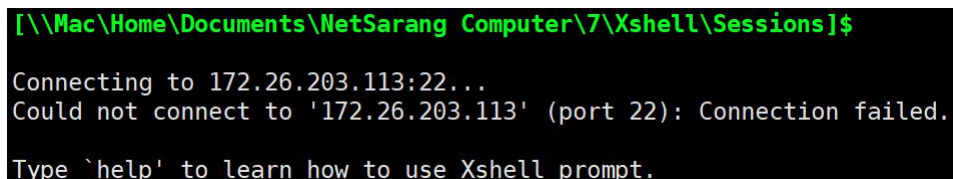
## 四、实验总结

### 4.1 总结上机实验过程中遇到的各类问题、困难以及解决过程中的收获。

本课程的实践主要是体验 GPU 对性能计算的提升，感受异构计算的意义所在。

在上个学期的并行计算课程中已经接触到了多线程计算 PI 值的几种算法并对其性能进行了分析，因此对算法的理解上并不存在问题。问题的关键在与掌握如何编写基于 CUDA 的 GPU 程序。

实验中遇到了无论如何连接不上会话的问题。

A screenshot of an Xshell terminal window. The title bar shows the path: [\\Mac\Home\Documents\NetSarang Computer\7\Xshell\Sessions]\$ . The terminal text shows an attempt to connect to 172.26.203.113:22... which fails with the message: 'Could not connect to '172.26.203.113' (port 22): Connection failed.' It then prompts the user to type 'help' to learn how to use the Xshell prompt.

```
[\\Mac\Home\Documents\NetSarang Computer\7\Xshell\Sessions]$  
Connecting to 172.26.203.113:22...  
Could not connect to '172.26.203.113' (port 22): Connection failed.  
Type 'help' to learn how to use Xshell prompt.
```

助教帮忙解决，原来是学校因停电导致服务器重启，其 IP 和端口重新分配。

对幂级展开编程时，对项的正负曾一度因为对 CUDA 的 block 和 thread 的理解未能成功处理，最后在代码中解决。

### 4.2 对基于 CUDA 的 GPU 计算的理解与分析。

在实验的过程中，我对老师上课时讲的 CUDA 函数和它的约束有了进一步的理解，体会到了内核分配线程数，块数的作用，也进一步学习了 CUDA 编程的几个具体步骤，如一开始分配 host 内存，对数据进行初始化，再到分配 device 内存，从 host 将数据拷贝到 device，再到调用 CUDA 核函数在 device 上计算，最后将 device 上的结果拷回到 host 上，释放 device，host 释放内存空间并由 host 归一处理数据。

---

我们可以从实验结果比较中发现积分法和幂级展开法效果相近，都可以看出 GPU 的计算性能的显著提高。面对类型高度统一的，相互无依赖的大规模数据和不需要打断的计算环境，GPU 展现出了高效的处理并行任务的能力。例如在积分法计算 PI 程序中，因为其中的矩阵法则将计算分成了多块，而这些计算过程高度相似并且计算过程之间并没有依赖的关联，所以可以分配 GPU 完成计算。

五、课程总结

附：上机实验与课程知识点分析

序号	上机实验内容	理论知识点	分析总结
1	基于CUDA的GPU计算PI值	使用 GPU 计算 PI 值 使用两种方法—— 积分法和幂级展开法	在代码中均已经实现。个人心得见上文。
2			
3			