****

《异构计算》



**实验1：基于 CUDA 的 GPU 计算 PI 值**

**学 院 智能与计算学部**\_\_

**专 业 计算机科学与技术**

**年 级 2019\_\_\_\_\_\_\_**

**姓 名 张明君（留学生）**

**学 号 6319000359\_\_\_\_\_\_**

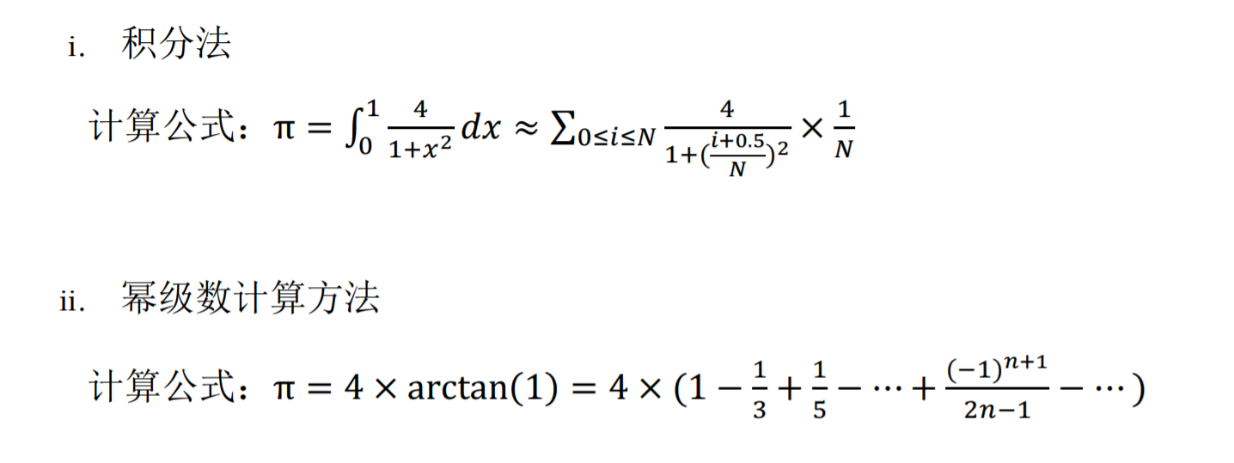
**2021年 9 月 30 日**

1. **实验内容**

本次实验使用 CUDA 编程方法来计算 pi 的值，分别使用积分法和幂级数计算方法来计算 pi 的值，并通 过 CUDA 修改 GPU 参数来测试性能，并同串行计算的性能比较计算加速比。

1. **实验原理**

本次实验 要求我们使用 CUDA 编程来 实现2种算法（积分方法和幂级数计算方法）来计算PI值：



1. **程序流程图**

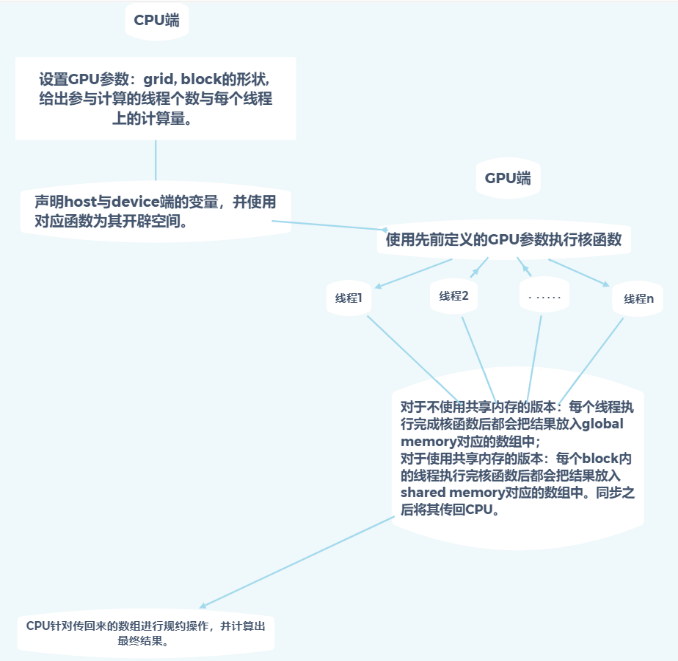
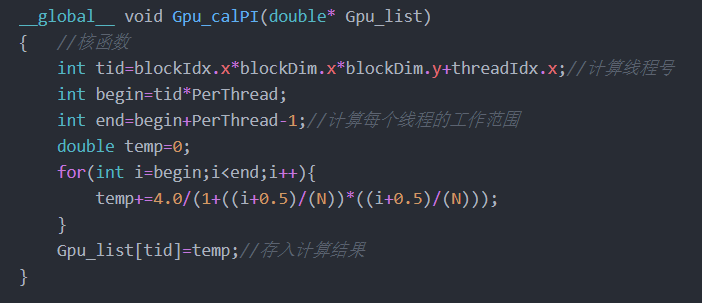
****

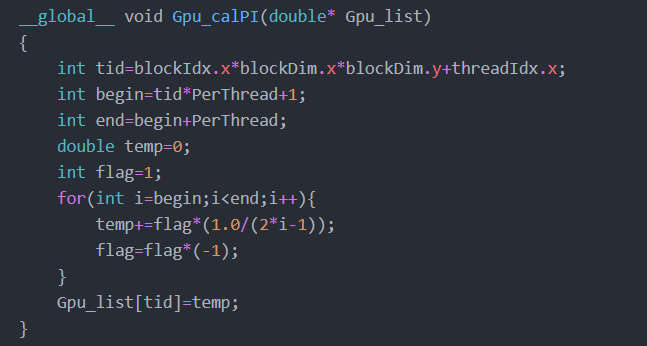
Figure 1：程序执行的流程图

通过使用GPU的block/thread等参数的调节纵向对比两种异构算法的性能；通过是否使用shared memory来对比共享内存对程序性能的影响；通过将异构并行程序与串行程序横向比较，分析异构计算对算法性能的提升，并计算出加速比与效率，绘制相对的曲线。给出实验结果与分析。

实现过程中分别给出了线性代码、CUDA异构使用共享内存版本 和不使用共享内存代码。

积分方法 代码实现求 PI值：



积分方法和幂级数计算方法 代码实现求 PI值：

1. **实验结果及分析**
2. **积分积分方法结果与分析**

* 在线程规模相同，但总计算量N不同的情况下，花费时间为：

下面的异构程序总共使用64个block，每个block中256个threads。

我们可以建一个数据表 如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法（N次数） | 16777216 | 16777216\*2 | 16777216\*4 | 16777216\*8 | 16777216\*16 |
| 线性计算时间 | 0.409154 | 0.733863 | 1.404614 | 2.83981 | 5.659504 |
| 异构计算实间 | 0.000729 | 0.001209 | 0.002084 | 0.003770 | 0.007211 |
| 异构计算实间（Share Memory） | 0.000624 | 0.001052 | 0.001876 | 0.003645 | 0.007084 |

加速比为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法（N次数） | 16777216 | 16777216\*2 | 16777216\*4 | 16777216\*8 | 16777216\*16 |
| 异构计算 | 561.25 | 607.08 | 673.99 | 753.26 | 784.84 |
| 异构计算（Share Memory） | 655.69 | 697.68 | 748.72 | 779.09 | 798.91 |

Figure 2：线程规模相同, N不同时加速比

* 在总计算量N相同的情况下，不同的block\*thread异构计算花费时间为：

我们使用N=67108864,此时线性算法运行时间为1.404614

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Block\*Thread | 32\*64 | 32\*128 | 32\*256 | 64\*64 | 64\*128 | 64\*256 |
| 异构计算时间 | 0.008341 | 0.004468 | 0.002578 | 0.004556 | 0.002545 | 0.002110 |
| 异构计算时间（Share Memory） | 0.008457 | 0.004457 | 0.002470 | 0.004495 | 0.002487 | 0.001907 |

加速比为:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Block\*Thread | 32\*64 | 32\*128 | 32\*256 | 64\*64 | 64\*128 | 64\*256 |
| 异构计算 | 168.39 | 314.37 | 544.84 | 308.29 | 551.91 | 665.69 |
| 异构计算（Share Memory） | 166.08 | 315.14 | 568.66 | 312.48 | 564.78 | 736.55 |

Figure 3：N相同，线程规模不同时的加速比

从上面的数据表 和加速比曲线 可以看到 CUDA 异构计算共享内存版本的加速比 总是大于 不使用共享内存的版本。随着计算量的增长，读出来的时间很长，运行出来的答案也会很长。

从图3我们可以看出，使用共享内存版本的加速比始终比另一版本高。且由表格数据可以发现在线程总数不变的情况下，block数量的增加会使得加速比略微下降，这是因为每个block共享一个共享内存，block数目增多导致等待同步的次数增多，导致加速比略微减小。

1. **幂级数计算方法**

* 在线程规模相同，但总计算量N不同的情况下，花费时间为：

异构程序总共使用64个block，每个block中256个threads。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法（N次数） | 16777216 | 16777216\*2 | 16777216\*4 | 16777216\*8 | 16777216\*16 |
| 线性计算时间 | 0.126978 | 0.204826 | 0.349940 | 0.668813 | 1.283428 |
| 异构计算实间 | 0.000619 | 0.0009230 | 0.001441 | 0.002551 | 0.004793 |
| 异构计算实间（Share Memory） | 0.000442 | 0.000741 | 0.001324 | 0.002403 | 0.004622 |

该情况下加速比为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法（N次数） | 16777216 | 16777216\*2 | 16777216\*4 | 16777216\*8 | 16777216\*16 |
| 异构计算 | 205.13 | 221.91 | 242.84 | 262.17 | 267.71 |
| 异构计算（Share Memory） | 287.28 | 276.41 | 264.30 | 278.32 | 277.67 |

Figure 4：线程规模相同, N不同时加速比

* 在总计算量N相同的情况下，不同的block\*thread异构计算花费时间为：

使用了N=67108864,此时线性算法运行时间为0.349940 为主。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Block\*Thread | 32\*64 | 32\*128 | 32\*256 | 64\*64 | 64\*128 | 64\*256 |
| 异构计算时间 | 0.005591 | 0.003021 | 0.001799 | 0.003033 | 0.001774 | 0.001478 |
| 异构计算时间（Share Memory） | 0.005585 | 0.002949 | 0.001670 | 0.002972 | 0.001673 | 0.001230 |

加速比为:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Block\*Thread | 32\*64 | 32\*128 | 32\*256 | 64\*64 | 64\*128 | 64\*256 |
| 异构计算 | 62.58 | 115.83 | 194.51 | 115.37 | 197.26 | 236.76 |
| 异构计算（Share Memory） | 62.65 | 118.66 | 209.54 | 117.74 | 209.16 | 284.50 |

Figure 5：N相同，线程规模不同时的加速比

在图4我们可以看出在线程规模一定的情况下，CUDA异构计算使用共享内存版本的加速比 竟然出现了 **上下波动趋势**，这可以证明 幂级数计算法 在该阶段增加的时间多于串行算法，可能导致该情况出现的原因，我觉得 是因为线程调度的问题导致同步时间增加，但是 总体趋势高于不使用共享内存的版本。

到了图5我们也可以来看，CUDA异构计算使用共享内存的加速比仍然 比另一个版本高。而且 在线程总数不变的情况下，block数量的增加会使得加速比略微下降，这与使用积分算法 得出的结论一致。

从上面的数据表和图 幂级数算法的性能都优于积分法的性能，并且在相同的算法中，使用共享内存的版本往往比 未使用的版本性能更好，这是因为 与全局内存（Global Memory）相比，从共享内存（Share Memory）访问数据花费的时间更少，所以程序的性能更好。但由于需要同步等机制，当线程规模较大时，这将成为多线程程序的瓶颈。

相比于串行程序，随着计算量的增大，异构并行程序的加速比都在提升。其原因是因为随着计算量的增大，GPU内部对线程调度产生的非计算时间所占的比例减小，而单个线程计算的时间增加，这反而会让程序整体用于计算的时间比例增大，因此相比于串行程序，异构并行计算的性能反而有所提升。

1. **实验总结**

本次实验是第一次编写CUDA程序。虽然课上已经学习了相关语法，而且在上学期也学习过并行计算课程。但第一次写还是有些无法适应。主要是对于CUDA中线程的组织层次了解不清晰，通过查阅资料，最终成功完成了程序的编写。

在查阅资料的过程中发现了共享内存这个概念，这正好解释了我之前一直疑惑的问题：每个线程都对传入的那个对象访问，效率可能会很低。于是我就开始编写了两个使用共享内存版本的程序。刚开始时没有使用同步策略，导致结果很奇怪。在查阅资料后完成编程。

本次实验 让我更了解 GPU 的好处 和瓶颈，共享内存 和没有共享内存 的效率差别。我希望下次实验能让我更深地了解CUDA 和 OpenCL，了解到了使用 CUDA 充分利用 GPU 性能能够大幅提升可并行化程序的性能。