**《并行计算》课程实验报告**

**实验4：基于GPU加速器的异构并行编程**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 常添 | | 院系 | | 计算学部 | | | 学号 | | 2022111699 | |
| 任课教师 | | 张伟哲 | | | | 指导教师 |  | | | | |
| 实验地点 | | 格物楼213 | | | | 实验时间 | 2024年10月18日 | | | | |
| 实验课表现 | | 出勤、表现得分 | |  | | 实验报告  得分 |  | | 实验总分 | |  |
| 操作结果得分 | |  | |
| **一、实验目的** | | | | | | | | | | | |
| 本次实验的主要目的包括以下几点：  1. 熟悉远程访问服务器和GPU加速计算环境：通过使用远程工具（如XSHELL和XFTP）连接并访问配有NVIDIA GPU的服务器，了解如何检查GPU状态及验证GPU可用性。  2. 掌握CUDA并行编程的基础原理与执行过程：通过编写、编译和运行CUDA程序（如矢量加法和矩阵乘法），理解CUDA编程的基本构造，如线程、线程块和线程格的概念。  3. 优化基于GPU的并行计算：通过实验不同线程块尺寸，分析CUDA程序的性能表现，理解CUDA优化技术在科学计算中的应用。  如何实现这些目的：  1. 远程连接和GPU验证：  - 使用XSHELL和XFTP工具远程连接服务器，熟悉远程环境的配置和文件管理操作。通过命令nvidia-smi查看GPU的状态，确认GPU可用并获取设备的基本信息，如温度、风扇转速、内存使用率等。  2. CUDA编程实践：  - 通过编写并运行一个简单的矢量加法程序（VectorAdd.cu），学习CUDA的编程模型，包括如何申请GPU内存、如何传输数据，以及如何编写核函数执行并行计算。  - 在执行CUDA程序的过程中，通过不同规模的线程和线程块配置，体验并分析并行计算在GPU上的效率表现。  3. CUDA优化与性能分析：  - 针对矩阵乘法的优化，实验中通过调整线程块的尺寸，记录程序的运算时间和运算速度。通过这些实验数据，计算出运算速度的提升比，分析不同配置对性能的影响。 | | | | | | | | | | | |
| **二、实验内容** | | | | | | | | | | | |
| 在本次实验中，我完成了以下工作：  1. 远程连接GPU服务器：  - 使用XSHELL和XFTP工具，通过给定的服务器信息，成功连接到Nvidia RTX 2080Ti和Tesla V100S GPU服务器。  - 通过命令nvidia-smi验证了GPU的可用性，并查看了设备的状态信息，包括GPU利用率、显存使用情况、温度等。  2. 编写和运行CUDA矢量加法程序：  - 编写了名为VectorAdd.cu的CUDA程序，实现两个512元素的整数数组的并行相加。  - 在主机端申请并初始化数组，使用cudaMalloc申请GPU内存，随后使用cudaMemcpy将数据从主机传输到设备端。  - 编写核函数，利用CUDA的线程机制并行执行加法操作。每个线程负责对应数组中的一个元素。  - 使用cudaMemcpy将结果从设备端传回主机端，并在主机端输出结果验证程序正确性。  - 通过nvcc命令编译程序，并成功运行，输出了每个元素的加法结果。  3. 基于CUDA的矩阵乘法优化实验：  - 编写和运行了矩阵乘法的CUDA程序，实验中采用了不同的线程块尺寸配置，观察并记录了程序在不同尺寸下的运算时间和性能表现。  - 利用cudaMallocPitch函数分配设备内存，使用cudaMemcpy2D函数传输二维数组数据，并通过核函数实现矩阵乘法的并行计算。  - 比较了不同线程块尺寸下的运算速度和时间，完成了多次测试，记录了各个配置的实验数据。  4. 性能分析与数据记录：  - 记录了不同线程块尺寸下的运算时间和速度，并根据数据计算了运算速度提升比（GFLOPS提升比）。  - 通过不同矩阵尺寸的测试，观察到不同线程块配置对于计算性能的影响。  - 在完成实验后，绘制了运算速度提升比的图表，并分析了不同配置的性能差异。  这些工作帮助我完成了对GPU并行编程的实践，深入理解了CUDA的执行过程和优化方法。 | | | | | | | | | | | |
| **三、实验结果** | | | | | | | | | | | |
| 首先检查服务器GPU是否可用，截图如下：  编译并运行 VectorAdd.cu，记录实验结果。    电脑萤幕画面  低可信度描述已自动生成  编译并运行 MatrixMul.cu    运行程序，进行如下实验并记录数据（实验报告中给出数据并绘图）  实验一：设置矩阵尺寸为，调整线程块尺寸，记录程序运算时间及运算速度。  16    32    64    128    256    512    基准运行时间(s)：0.070592  基准运算速度(GFLOPS)：0.44   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | NUM\_THREADS | 16 | 32 | 64 | 128 | 256 | 512 | | 运算时间(s) | 0.115111 | 0.097290 | 0.104160 | 0.110152 | 0.106105 | 0.104895 | | 运算速度(GFLOPS) | 0.27 | 0.32 | 0.30 | 0.28 | 0.29 | 0.30 |   实验二：设置矩阵尺寸为，调整线程块尺寸，记录程序运算时间及运算速度。  16    32    64  图形用户界面, 文本, 网站  描述已自动生成  128    256    512    基准运行时间(s)： 0.550010  基准运算速度(GFLOPS)：0.45   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | NUM\_THREADS | 16 | 32 | 64 | 128 | 256 | 512 | | 运算时间(s) | 0.105059 | 0.095922 | 0.107438 | 0.104731 | 0.107081 | 0.108796 | | 运算速度(GFLOPS) | 2.38 | 2.61 | 2.33 | 2.39 | 2.33 | 2.30 |   实验三：设置矩阵尺寸为，调整线程块尺寸，记录程序运算时间及运算速度：  16    32    64    128    256    512    基准运行时间(s)：4.195637  基准运算速度(GFLOPS)：0.48   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | NUM\_THREADS | 16 | 32 | 64 | 128 | 256 | 512 | | 运算时间(s) | 0.100537 | 0.122445 | 0.091129 | 0.092591 | 0.099906 | 0.111145 | | 运算速度(GFLOPS) | 19.89 | 16.33 | 21.95 | 21.60 | 20.02 | 17.99 |   （3）根据记录的数据，计算运算速度提升比（GFLOPS的比值）并绘图  实验一：设置矩阵尺寸为   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | NUM\_THREADS | 16 | 32 | 64 | 128 | 256 | 512 | | 运算速度提升比 | 0.614 | 0.727 | 0.682 | 0.636 | 0.659 | 0.682 |   实验二：设置矩阵尺寸为   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | NUM\_THREADS | 16 | 32 | 64 | 128 | 256 | 512 | | 运算速度提升比 | 5.289 | 5.800 | 5.178 | 5.311 | 5.178 | 5.111 |   实验三：设置矩阵尺寸为   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | NUM\_THREADS | 16 | 32 | 64 | 128 | 256 | 512 | | 运算速度提升比 | 41.438 | 34.021 | 45.729 | 45.000 | 41.708 | 37.479 |   绘制不同矩阵规模下NUM\_THREADS和运算速度提升比的关系图像如下： | | | | | | | | | | | |
| **四、思考题** | | | | | | | | | | | |
| 思考题1：查阅相关资料，对实验所得的加速比图进行分析。  实验加速比图的分析：  从图中可以看出，三个子图分别对应不同的矩阵大小（250x250、500x500、1000x1000），并且每个子图显示了不同线程块尺寸下的加速比（Speedup）变化趋势。根据这些数据可以进行如下分析：  1. 矩阵尺寸为250x250时：  - 在线程数为32时，加速比最高达到0.73。随着线程数的增加，加速比先明显下降，随后有小幅回升。  - 小尺寸矩阵的加速效果相对有限，可能是因为线程过多导致资源分配不合理，出现了性能下降。  2. 矩阵尺寸为500x500时：  - 当线程数为32时，加速比达到最高点5.80，说明此时的线程块配置较为合适。随着线程块尺寸的增大，加速比呈现先下降后持平的趋势。  - 对于中等大小的矩阵，线程块尺寸的合理配置对性能提升有显著作用。线程数过大时，由于线程调度和内存带宽的瓶颈，性能未能继续提升。  3. 矩阵尺寸为1000x1000时：  - 对于大矩阵，加速比在线程数为32时达到最大值45.73，随后随着线程数的增大，加速比持续下降，表明过多的线程反而导致了性能损失。  - 这可能是由于大矩阵计算时，更多的线程块需要处理更多数据，而线程块之间的同步和内存带宽使用加剧了资源竞争，最终导致性能降低。  综合分析：  - 通过对比不同矩阵尺寸下的加速比，可以得出结论：线程块尺寸的选择对于GPU并行计算性能具有重要影响。尤其是当矩阵尺寸较小时，过大的线程块反而会导致资源浪费和性能下降。  - 最佳线程块尺寸在这三个实验中均出现在线程数为32时，说明在实际计算中，32线程块的配置能够在大多数情况下达到较好的性能。  - 大矩阵的加速比明显高于小矩阵，表明GPU加速在处理大规模计算时优势更明显。但与此同时，线程块的合理分配和同步机制也变得更加重要。 | | | | | | | | | | | |
| **五、实验心得体会** | | | | | | | | | | | |
| 通过本次实验，我对GPU并行编程和CUDA的工作原理有了更加深入的理解，主要体会如下：  1. GPU并行计算的优势：  - GPU在处理大规模并行计算任务时，能够显著提高计算效率。通过本次实验，我切身体会到，当矩阵规模增大时，GPU的并行计算能力可以实现数倍甚至几十倍的加速效果。尤其是在矩阵乘法的实验中，GPU的计算速度相比于CPU展现了明显的优势。  2. 线程块配置对性能的影响：  - 实验中，不同线程块尺寸对程序的性能产生了较大影响。合理的线程块配置能够充分利用GPU资源，达到最佳的性能提升效果。过大的线程块会导致资源浪费和性能瓶颈，反而降低了效率。通过对加速比的分析，我认识到根据不同问题规模选择合适的线程块配置非常重要。  3. CUDA编程实践：  - 通过编写和运行CUDA程序，我熟悉了CUDA的基本编程模型，如线程、线程块、线程格的概念，以及CUDA的内存管理方式。在程序中，我学习了如何通过cudaMalloc分配GPU内存、使用cudaMemcpy在主机与设备之间传递数据，并通过核函数实现并行计算。  - 这些操作让我加深了对GPU并行计算机制的理解，尤其是在数据传输和内存管理的过程中，我学会了如何避免常见的性能瓶颈，比如减少内存访问次数、优化内存布局等。  4. 实验问题与解决：  - 在实验过程中，我遇到了一些挑战，比如由于线程块设置不合理导致的性能下降，以及内存溢出等问题。在解决这些问题的过程中，我学会了通过调整线程块大小、优化代码结构，以及使用GPU的调试工具检查程序执行中的瓶颈。通过这些问题的解决，我对CUDA编程的细节和优化技巧有了更深入的理解。  5. 并行计算的实际应用：  - 通过本次实验，我进一步体会到GPU并行计算在科学计算、机器学习、图像处理等领域的巨大潜力。特别是对于大规模矩阵运算、图像处理等任务，GPU并行计算可以极大地提升计算效率。这次实验不仅让我学到了理论知识，也让我掌握了一些实际操作技能，能够将GPU计算应用到未来的研究和工作中。  总结：  这次实验不仅让我对CUDA并行编程有了实操经验，也让我理解了并行计算的优化思路与技巧。今后，我将更加注重如何优化程序的执行效率，尤其是在面对大规模数据集或高性能计算需求时，充分利用GPU的并行计算优势。 | | | | | | | | | | | |
| 指导教师评语：  日期： | | | | | | | | | | | |