高性能计算应用实践实验报告（lab9）

2024 秋 苏涵 2023311B25

实验环境

OS：Ubuntu 22.04.3 LTS
cpu 型号：11th Gen Intel(R) Core(TM) i7-11370H
频率：3.30GHz
核数：4
内存大小：16GB
显卡型号：NVIDIA RTX A2000 Laptop GPU

任务 1

1. 问题分析

通过不同的矩阵计算顺序和数据布局，可以改变性能。考虑的计算顺序包括：ijk、ikj、jik、jki、kij 和 kji。不同的顺序会影响缓存利用率、内存访问模式等，从而影响性能。此外，向量化 SIMD、矩阵分块和数据重排等技术也是优化 GEMM 性能的关键手段。

2. 方案设计及实现方法

本实验中，我在 how-to-optimize-gemm 框架添加了以下几种方案的 MY_MMult 函数

- 调整计算顺序：调换 for 循环语句顺序，编写 ijk、ikj、jki、jik、kij、kji 顺序的 GEMM 实现，并记录不同规模矩阵每种顺序的执行时间。
- 向量化 SIMD：利用 SIMD 指令集进行计算（用了 avx2），进行了向量化。
- 矩阵分块：由于 B 不是内存连续的，程序就要读取多次 B 矩阵的数据，这样数据存取将成为整个程序 gflops 上升的瓶颈。故将原始矩阵划分为 64x64 的小矩阵，分别计算后再合并结果，优化缓存使用，减少内存访问延迟。
- 数据重排：在分块后重新排列 A、B 矩阵数据，确保数据访问的连续性。

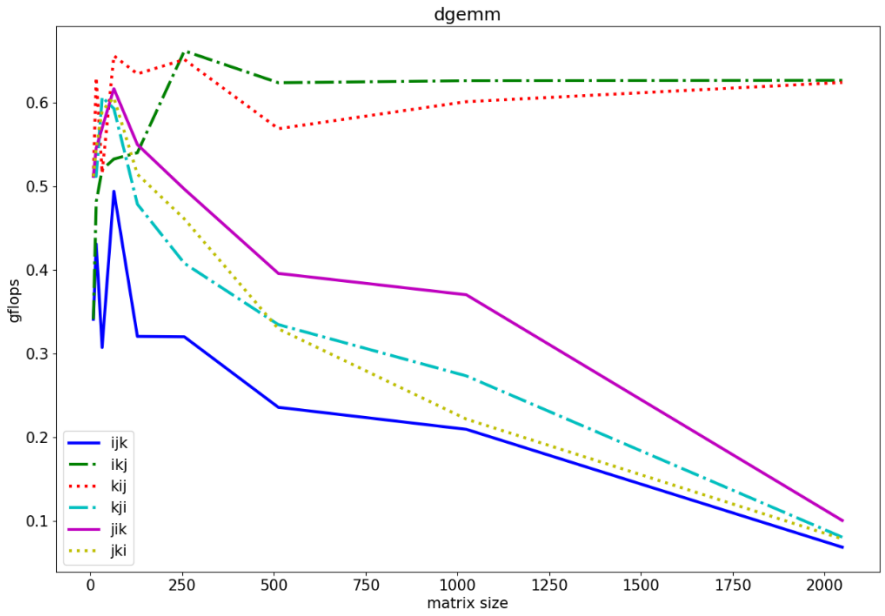
3. 结果分析

测量记录不同方案的 gflops 时间，结果如下：

GFLOPS	ijk	ikj	kij	kji	jik	jki
8	0.3413333	0.3413333	0.5120000	0.5120000	0.5120000	0.5120000
16	0.4311579	0.4818824	0.6301538	0.5120000	0.5461333	0.5461333
32	0.3076808	0.5201270	0.5160315	0.6068148	0.5698783	0.5904144
64	0.4941451	0.5328130	0.6561802	0.5930860	0.6168094	0.6061133

128	0.3208371	0.5402942	0.6345392	0.4789112	0.5500727	0.5147017
256	0.3203562	0.6620059	0.6517322	0.4081899	0.4968009	0.4612103
512	0.2361149	0.6239971	0.5690411	0.3349251	0.3961106	0.3299095
1024	0.2098076	0.6263835	0.6013171	0.2736344	0.3705864	0.2220228
2048	0.06895994	0.6267700	0.6242805	0.08090880	0.1009096	0.07886396

使用 python 将以上结果转化为可视化图像得:

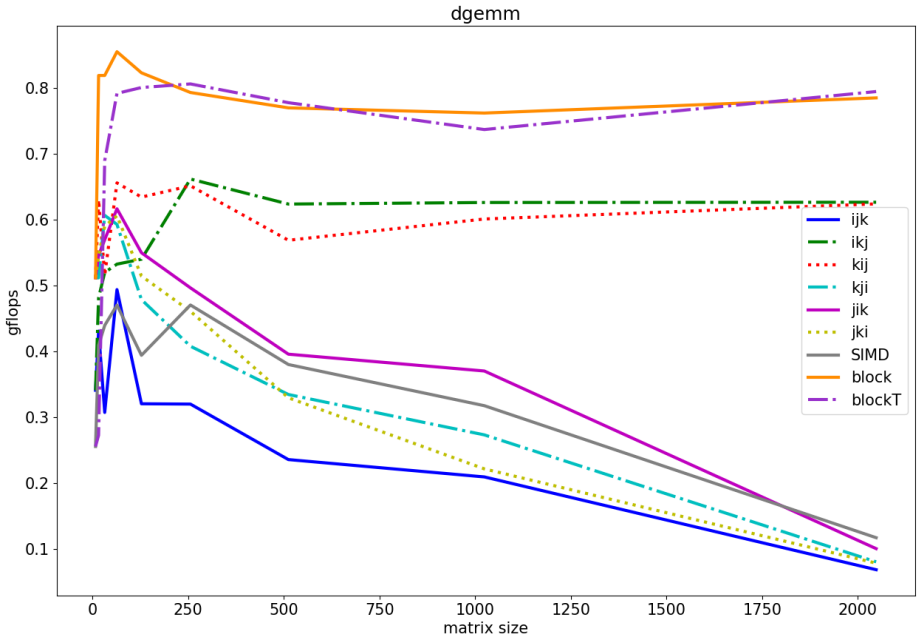


可见当矩阵规模小的时候，波动较大且各顺序之间无较明显性能差异，当矩阵规模增大，差异明显，性能从好到差依次为 ikj、kij、jik、kji、jki、ijk。且可从图中看出矩阵规模变大时，ikj、kij 顺序性能基本保持稳定，而其它则 gflops 明显变小。

GFLOPS	SIMD	分块	数据重排
8	0.2560000	0.5120000	0.2560000
16	0.4096000	0.8192000	0.2730667
32	0.4398389	0.8192000	0.6898526
64	0.4706355	0.8552822	0.7919758
128	0.3944239	0.8232196	0.8008982
256	0.4707808	0.7933428	0.8063449

512	0.3804336	0.7700253	0.7779137
1024	0.3179459	0.7621390	0.7371289
2048	0.1175286	0.7851604	0.7947936

使用 python 将以上所有方案结果转化为可视化图像得:



由图像可知，SIMD 向量化优化方法有一定效果，但在矩阵规模大时仍然会变差。而矩阵分块和分块后数据重排两种方案性能与其他方法相比均有明显优化，且在规模增大时仍能保持较高性能，而在规模小时，数据重排的 gflops 比未重排的还小，规模变大时两者十分相近。

任务 2

1. 问题分析

CPU 实现可以通过多种优化方法（例如任务 1）提高性能，而 GPU 由于其高度并行的计算架构，能够提供更优的性能。该任务将 GPU 的 GEMM 实现集成到 how-to-optimize-gemm 框架中，可利用 GPU 的计算能力进行优化。

2. 方案设计及实现方法

- **环境搭建**：安装 CUDA 工具包，确保 CUDA 环境正常运行，配置相应的编译器和库。
- **使用 CUDA 进行 GPU 编程**：利用 NVIDIA 的 CUDA 编程模型实现 GEMM，编写 MY_MMult 函数，充分利用 GPU 的并行计算能力。
- **集成到框架**：修改原有 makefile 文件，如使用 NVCC、增加-lcublas 等等（提交的代码中有注释标出修改部分）

3. 实现过程及遇到的问题

个人电脑有一张 Intel(R) Iris(R) Xe Graphics 和一张 NVIDIA RTX A2000 Laptop GPU 的独显，故使用个人电脑。

环境搭建：安装支持的 cuda 版本，由以下输出可知搭建完成

```
root@suh:~# nvidia-smi
Sat Oct  5 16:34:28 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.86.10              Driver Version: 537.58       CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA RTX A2000 Laptop GPU    On  | 00000000:F3:00.0 Off |                  N/A |
| N/A   41C    P8               4W /  33W |     81MiB /  4096MiB |      3%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

```
root@suh:~# nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Thu_Nov_18_09:45:30_PST_2021
Cuda compilation tools, release 11.5, V11.5.119
Build cuda_11.5.r11.5/compiler.30672275_0
```

核心代码实现过程及代码截图：

① 使用 cudaMalloc 在 GPU 设备上为 a、b、c 分配内存。

② 进行数据传输，使用 cudaMemcpy 将输入矩阵 a、b、c 复制到 GPU 上相应的内存位置。

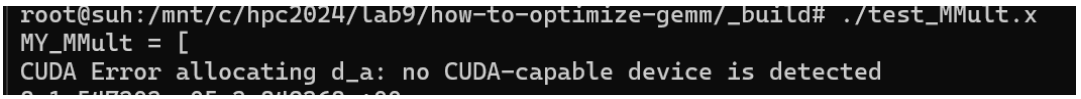
③ 使用 cublasDgemm 函数执行矩阵乘法。

④ 通过再次调用 cudaMemcpy 将计算结果从 GPU 内存的矩阵 c 复制回 CPU 内存。

⑤ 用 cudaFree 释放内存，同时，调用 cublasDestroy 来销毁 cuBLAS 句柄，释放相关资源。

问题:

编译通过，但运行时无法使用 GPU，显示 no CUDA-capable device is detected



上网搜集资料后尝试多种方法（修改 nvidia 控制面板全局、关掉 secure boot、使用 sudo prime-select nvidia、安装更低版本的 CUDA 等等）都无法正常运行。