高性能计算实验报告

实验3、5、6、9: 联合实验

2024秋季学期 姓名:曹馨尹 学号: 2023311708

一、 实验三: Linux环境下调试与矩阵乘法优化 (Openblas)

1. 多个 C 代码中有相同的 MY_MMult 函数,怎么判断可执行文件调用的是哪个版本的 MY_MMult 函数?是 Makefile 中的哪行代码决定的?

答: 可执行文件调用的 MY MMult 版本由 Makefile 中的链接顺序和目标文件的顺序决定。

2. 性能数据 _data/output_MMult0.m 是怎么生成的? C 代码中只是将数据输出到终端并没有写入文件。

答: 性能数据文件是通过重定向可执行文件的输出到文件完成的。

二、实验五: Linux环境多线程编程

1. 实验过程中自己认为值得记录的问题。

答: 暂无。

2. 测试较大规模的矩阵时cpu利用率和能体现多线程的运行截图,多线程查看使用top、ps、pstree等不限答:

top - 22:18:34 up 4:16, 1 user, load average: 0.05, 0.06, 0.02 1 running, 326 sleeping, 0 zombie Tasks: **327** total, 0 stopped, **0.7** sy, **0.0** ni, **95.4** id, **0.0** wa, 0.0 hi, %Cpu(s): **3.9** us, 0.0 si, MiB Mem : **3895.3** total, **346.6** free, **1913.5** used, **1958.2** buff/cache 1981.8 avail Mem MiB Swap: **3895.0** total, 3895.0 free, **0.0** used. PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 45740 11896 7627 superso+ 20 1408 S 46.8 0.3 0:01.41 test MM+ 0 20 0 1136.2g 299460 114728 S 9.6 7.5 3791 superso+ 2:28.42 code 2829 superso+ 20 0 4424080 287048 126452 S 7.6 7.2 0:47.66 gnome-s+ 4.7 3761 superso+ 20 0 32.5g 108112 95676 S 2.7 0:52.83 code 20 73036 S 3217 superso+ 0 309288 107176 2.0 2.7 0:18.80 Xwayland 7450 superso+ 20 0 738184 59720 46728 S 2.0 1.5 0:01.72 gnome-t+ 3694 superso+ 20 0 1134.1g 179704 127204 S 1.3 4.5 0:12.04 code 0:11.34 code 0 1134.0q 204896 77440 S 0.7 5.1 3836 superso+ 20 17 root 20 0 0 0 0 I 0.3 0.0 0:01.69 rcu_pre+ -51 0 S 620 root 0 0 0.3 0.0 0:00.38 irq/18-+ 0 388856 6784 S 0.3 0:00.49 ibus-da+ 2945 superso+ 20 0 11736 0.3 218224 3545 superso+ 20 0 3100 2688 S 0.3 0.1 0:06.32 VBoxCli+ 3553 superso+ 20 0 218740 2952 2688 S 0.3 0.1 0:15.25 VBoxCli+ 3847 superso+ 20 0 1134.0g 95248 71808 S 0.3 2.4 0:03.83 code 3906 superso+ 20 0 108180 45824 17792 S 0.3 1.1 0:08.02 cpptools 7182 root 20 0 0 0 0 I 0.3 0.0 0:00.44 kworker+ 7373 root 20 0 0 0 0 I 0.3 0.0 0:00.13 kworker+

三、实验六

汇总前面基于框架代码how-to-optimize-gemm不同方式的DGEMM实现,报告中要有以下内容:

1. 列出实验环境:操作系统版本,编译器版本,CPU的物理核数、频率

答:

OS版本: Ubuntu 24.04 LTS

GCC版本: 13.2.0

CPU 型号: AMD Ryzen 9 7940H

核数: 16 核

频率: BogoMIPS 为 7984.98 MHz **AVX 指令集版本:** 支持 AVX2 和 AVX

内存大小:

• 总内存 (Mem): 3.8 GiB

• 已用内存 (used): 1.1 GiB

• 空闲内存 (free): 2.2 GiB

• 共享内存 (shared): 32 MiB

• 缓冲/缓存 (buff/cache): 830 MiB

• 可用内存 (available): 2.7 GiB

2. 每种方式的简单介绍,实现的核心代码

```
答:
```

(1) lab3: 使用cblas函数,修改MMult_1.c文件。 #include <stdio.h> #include <cblas.h> /* Routine for computing C = A * B + C */ void MY_MMult(int m, int n, int k, double *a, int lda, double *b, int ldb, double *c, int ldc) { cblas dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m, n, k, 1.0, a, lda, b, ldb, 1.0, c, ldc); } (2) lab5: 使用结构体装载一个线程的所有信息 typedef struct { int m; // 矩阵 A 的行数 // 矩阵 B 的列数 int n; int k; // 矩阵 A 和 B 的共享维度 double *A; // 矩阵 A double *B; // 矩阵 B // 结果矩阵 C double *C; int lda; // 矩阵 A 的领先维度 int ldb; // 矩阵 B 的领先维度 int ldc; // 矩阵 C 的领先维度 int row start; // 当前线程处理的起始行 int row end; // 当前线程处理的结束行 } ThreadData; 让每个线程执行多线程函数 threaded mmult void *threaded_mmult(void *arg) { ThreadData *data = (ThreadData *)arg;}

MY_MMult 函数定义要使用的线程数量。根据指定的线程数量平均分配行,并为每个线程创建一个线程来执行矩阵乘法。

创建多个线程并分配任务

int rows_per_thread = m / num_threads;

修改makefile使其能够链接pthread库

LDFLAGS := -lm -lpthread

(3) lab6:

pragma omp parallel for 是一个 OpenMP 指令,用于指示编译器将后面的 for 循环并行化。它会创建一个线程池,多个线程将被分配来并行执行循环任务。

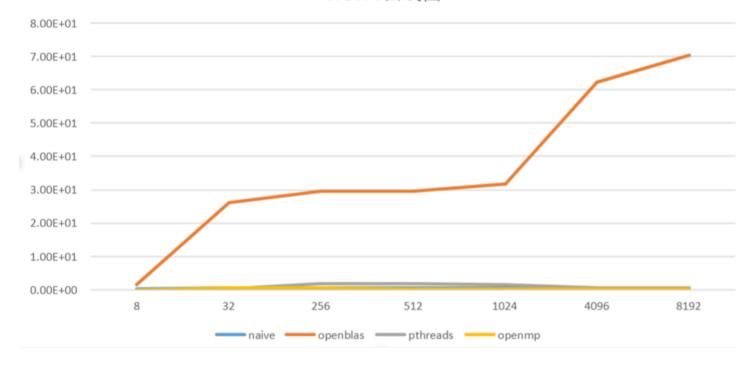
外层循环 for (i = 0; i < m; i++) 被并行化,表示每个线程可以同时处理不同的行。每个线程将独立计算 矩阵 C 的一行。每个线程会拥有一个私有的 j 和 p 变量,因为这些变量在各个线程之间独立使用。 private(j, p) 指定变量 j 和 p 为每个线程私有。

3. gflops曲线图, naive、openblas、pthreads、openmp四种情况画在一张图里面,矩阵规模至少包括8,32,256,512,1024,4096,8192。如果矩阵规模/CPU核心数不是整数,可以调整将矩阵规模调整,比如1000,4000。画图可以用plotAll.py或者excel,并对数据做汇总分析说明。

答:

矩阵大小	8	32	256	512	1024	4096	8192
naive	3.41E-01	6.07E-01	5.90E-01	5.89E-01	5.67E-01	1.76E-01	1.74E-01
openblas	1.62E+00	2.60E+01	2.94E+01	2.94E+01	3.17E+01	6.23E+01	7.02E+01
pthreads	5.49E-04	3.36E-01	1.66E+00	1.68E+00	1.51E+00	6.31E-01	6.02E-01
openmp	2.33E-02	5.12E-01	4.68E-01	3.83E-01	2.95E-01	2.77E-01	2.43E-01

GFLOPS曲线图



分析:

- naive 算法在小规模矩阵时效率低下,适合小规模数据的简单计算。
- openblas在中等规模时表现良好,适合需要优化的矩阵运算。
- pthreads 和 openmp适合大规模矩阵,适合需要高性能计算的场景。但这一点在图中没有明显反映出来。原因可能是在并行计算中,线程的创建、销毁和同步都需要一定的时间。在小规模矩阵时,这些开销可能占用的比例较大,导致GFLOPS并没有显著提高。而在大规模矩阵时,如果算法在计算上并没有显著的提升,仅仅是由于线程的高开销,整体性能可能看起来不会有预期的提升。
- 4. 开启openmp时cpu利用率和能体现OpenMP开启多个线程的运行截图,OpenMP碰到的问题及解决过程,前面lab3和lab5要求记录和回答的内容。

答:

```
top - 22:26:54 up 4:24, 1 user, load average: 0.14, 0.05, 0.01
Tasks: 339 total,
                    3 running, 336 sleeping,
                                                0 stopped,
                   0.2 sy, 0.0 ni, 95.0 id,
%Cpu(s):
          4.7 us,
                                               0.0 wa, 0.0 hi,
                                                                 0.0 si,
MiB Mem :
            3895.3 total,
                            350.7 free,
                                            1907.7 used,
                                                           1957.5 buff/cache
                                                           1987.6 avail Mem
MiB Swap:
            3895.0 total,
                            3895.0 free.
                                               0.0 used.
                                            SHR S
    PID USER
                  PR NI
                            VIRT
                                    RES
                                                  %CPU %MEM
                                                                  TIME+ COMMAND
   7834 superso+
                           44144
                                  34812
                                           1792 R 100.0
                  20
                       0
                                                          0.9
                                                                0:06.18 test_MM+
   2829 superso+
                  20
                       0 4425924 288060 126620 R
                                                    6.6
                                                          7.2
                                                                0:57.28 gnome-s+
   3791 superso+
                  20
                       0 1134.1g 282500 111580 S
                                                    3.0
                                                          7.1
                                                                2:30.10 code
   3761 superso+
                  20
                      0
                           32.5g 103740 91340 S
                                                    2.3
                                                          2.6
                                                                0:54.76 code
                  20
                       0
                          309288 107176
                                          73036 S
                                                    1.3
                                                          2.7
                                                                0:19.80 Xwayland
   3217 superso+
                       0 738940 60860
   7450 superso+
                  20
                                         46972 S
                                                    1.3
                                                          1.5
                                                                0:03.66 gnome-t+
                  20
                       0 1134.0g 205084
                                          77440 S
                                                    1.0
                                                          5.1
                                                                0:11.70 code
   3836 superso+
   3536 superso+
                  20
                       0 878860 103416
                                          77220 S
                                                    0.7
                                                          2.6
                                                                0:02.84 mutter-+
   3694 superso+
                  20
                       0 1134.1g 180528 127204 S
                                                    0.7
                                                          4.5
                                                                0:12.52 code
                                           2688 S
                                                                0:16.27 VBoxCli+
   3553 superso+
                  20
                       0 218740
                                    2952
                                                    0.3
                                                          0.1
                                         71808 S
   3847 superso+
                  20
                       0 1134.0g
                                  95376
                                                    0.3
                                                          2.4
                                                                0:04.05 code
                                                                0:08.17 cpptools
                  20
                       0
                          108692
                                  46336
                                         17792 S
                                                    0.3
                                                          1.2
   3906 superso+
   7835 superso+
                  20
                       0
                           14504
                                   5888
                                           3712 R
                                                    0.3
                                                          0.1
                                                                0:00.03 top
                                           9304 S
      1 root
                  20
                       0
                           23436
                                  13912
                                                    0.0
                                                          0.3
                                                                0:13.34 systemd
      2 root
                  20
                       0
                               0
                                       0
                                              0 S
                                                    0.0
                                                          0.0
                                                                0:00.03 kthreadd
      3 root
                  20
                       0
                               0
                                       0
                                              0 S
                                                    0.0
                                                          0.0
                                                                0:00.00 pool wo+
                                              0 I
      4 root
                   0 -20
                               0
                                       0
                                                    0.0
                                                          0.0
                                                                0:00.00 kworker+
                   0 - 20
                               0
                                       0
                                              0 I
                                                    0.0
                                                          0.0
                                                                0:00.00 kworker+
      5 root
```

四、实验九

在how-to-optimize-gemm框架进一步探索单核的优化方法,包括调整ijk顺序、向量化SIMD、矩阵分块、分块后数据重排,ijk顺序组合有:ijk,ikj, jik, jki, kij, kji

矩阵大小	8	32	256	512	1024
优化方法					
ijk	5.12E-01	5.80E-01	4.69E-01	3.82E-01	3.72E-01

矩阵大小	8	32	256	512	1024
ikj	3.41E-01	6.07E-01	5.90E-01	5.89E-01	5.67E-01
jik	5.12E-01	4.46E-01	4.50E-01	3.77E-01	3.66E-01
jki	5.12E-01	3.49E-01	3.49E-01	4.15E-01	2.56E-01
kij	5.12E-01	4.40E-01	6.00E-01	5.90E-01	5.88E-01
kji	5.12E-01	6.18E-01	4.59E-01	3.22E-01	1.89E-01

