

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
Tasks: 327 total, 1 running, 326 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.9 us, 0.7 sy, 0.0 ni, 95.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3895.3 total, 346.6 free, 1913.5 used, 1958.2 buff/cache
MiB Swap: 3895.0 total, 3895.0 free, 0.0 used. 1981.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
7627	superso+	20	0	45740	11896	1408	S	46.8	0.3	0:01.41	test_MM+
3791	superso+	20	0	1136.2g	299460	114728	S	9.6	7.5	2:28.42	code
2829	superso+	20	0	4424080	287048	126452	S	7.6	7.2	0:47.66	gnome-s+
3761	superso+	20	0	32.5g	108112	95676	S	4.7	2.7	0:52.83	code
3217	superso+	20	0	309288	107176	73036	S	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
3836	superso+	20	0	1134.0g	204896	77440	S	0.7	5.1	0:11.34	code
17	root	20	0	0	0	0	I	0.3	0.0	0:01.69	rcu_pre+
620	root	-51	0	0	0	0	S	0.3	0.0	0:00.38	irq/18+
2945	superso+	20	0	388856	11736	6784	S	0.3	0.3	0:00.49	ibus-da+
3545	superso+	20	0	218224	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
    int n;           // 矩阵 B 的列数
    int k;           // 矩阵 A 和 B 的共享维度
    double *A;       // 矩阵 A
    double *B;       // 矩阵 B
    double *C;       // 结果矩阵 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：(暂缺，周五晚之前补上)

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 zombie
%Cpu(s): 4.7 us, 0.2 sy, 0.0 ni, 95.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3895.3 total, 350.7 free, 1907.7 used, 1957.5 buff/cache
MiB Swap: 3895.0 total, 3895.0 free, 0.0 used. 1987.6 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
7834	superso+	20	0	44144	34812	1792	R	100.0	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
3217	superso+	20	0	309288	107176	73036	S	1.3	2.7	0:19.80	Xwayland
7450	superso+	20	0	738940	60860	46972	S	1.3	1.5	0:03.66	gnome-t+
3836	superso+	20	0	1134.0g	205084	77440	S	1.0	5.1	0:11.70	code
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
3553	superso+	20	0	218740	2952	2688	S	0.3	0.1	0:16.27	VBoxCli+
3847	superso+	20	0	1134.0g	95376	71808	S	0.3	2.4	0:04.05	code
3906	superso+	20	0	108692	46336	17792	S	0.3	1.2	0:08.17	cpptools
7835	superso+	20	0	14504	5888	3712	R	0.3	0.1	0:00.03	top
1	root	20	0	23436	13912	9304	S	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+
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker+
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker+

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

[illegible]

优化方法	矩阵大小	矩阵大小	矩阵大小	矩阵大小
ijk				
ijk				
ijk				
ijk				
ijk				
数据还没跑完 ()				