lab5-Linux环境多线程编程

朱宏基　　220110131　　大二(上)

实验方案:

首先明确我们的目标是利用多核处理器并行运算来加速矩阵相乘的计算及通过矩阵分块减少内存访问的开销,接着根据目标尝试设计适用的函数实现相应功能:
设计了 `dgemm_block` 函数执行一个块的矩阵乘法,并将结果累加到全局矩阵C中。
并且设计了调用 `dgemm_block` 函数的 `dgemm_thread` 函数,这个函数通过将矩阵相乘的工作分配给多个线程,每个线程负责计算一部分块的乘法,从而在多核处理器上同时执行多个乘法操作,从而提高性能。
而在主线程 `main` 中,初始化了矩阵A和B,并创建多个线程,等待其完成工作,同时跟踪计算了 `GFLOPS` 。

实验环境:

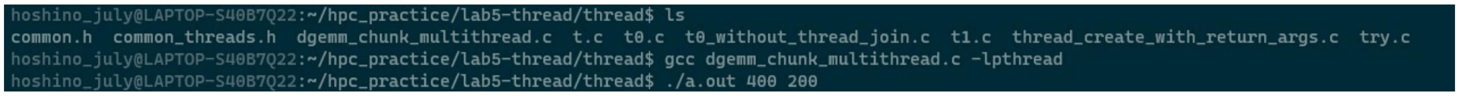
- **OS版本**: Ubuntu 22.04.2 LTS
- **gcc版本**: 11.4.0
- **CPU**:
    i. 型号: AMD Ryzen 5 4600H with Radeon Graphics
    ii. 频率: 均为 2994.389 MHz
    iii. 核数: 12
- **内存**: 7.5 Gi

关键代码:

```
// Function to perform DGEMM block multiplication
void dgemm_block(const int i, const int j, const int k)
{
    for (int x = i; x < i + block_size; x++)
    {
        for (int y = j; y < j + block_size; y++)
        {
            for (int z = k; z < k + block_size; z++)
            {
                C[x * lda + y] += A[x * lda + z] * B[z * lda + y];
            }
        }
    }
}

// Thread function for parallel DGEMM
void *dgemm_thread(void *arg)
{
    long thread_id = (long)arg;
    int blocks_per_thread = lda / block_size / num_threads;
    int start_block = thread_id * blocks_per_thread;
    int end_block = (thread_id == num_threads - 1) ? (lda / block_size) : (start_block + blocks_

    for (int i = 0; i < lda; i += block_size)
    {
        for (int j = 0; j < lda; j += block_size)
        {
            for (int k = 0; k < lda; k += block_size)
            {
                dgemm_block(i, j, k);
            }
        }
    }

    pthread_exit(NULL);
}
```

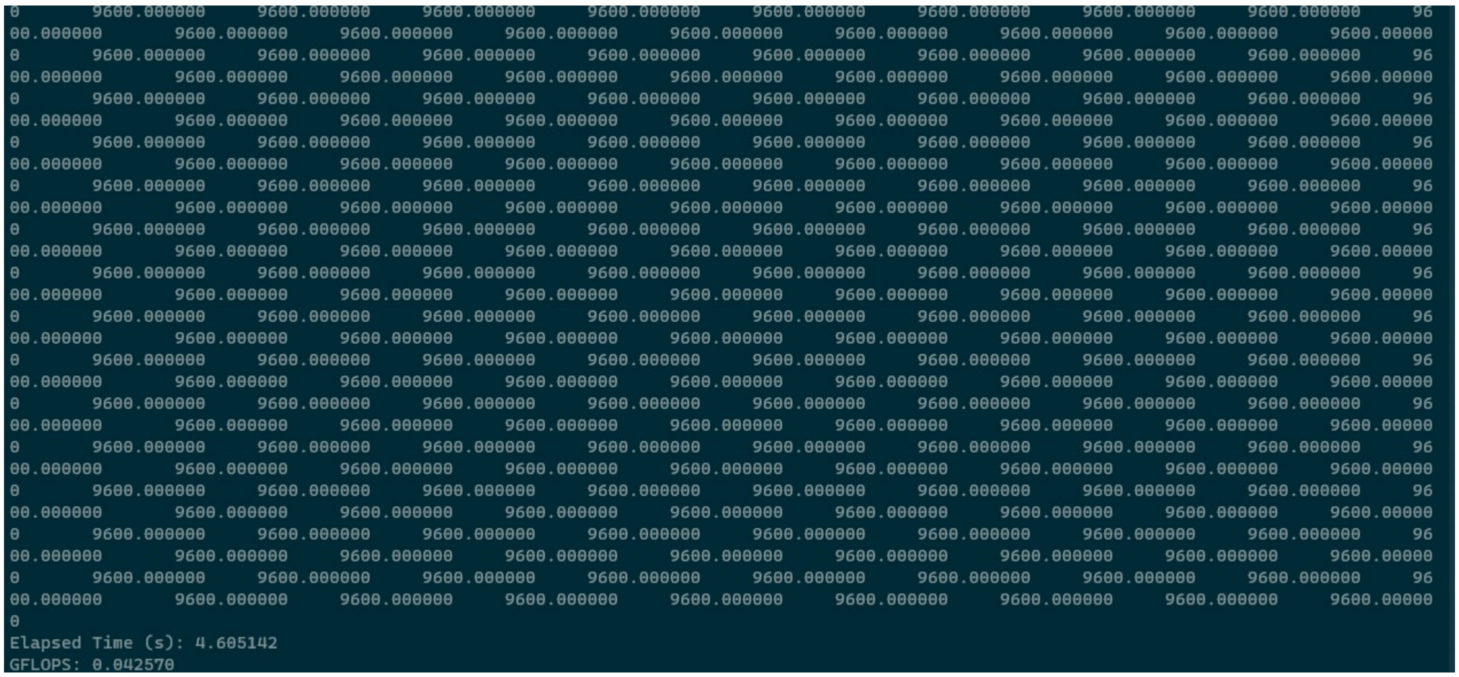
运行结果:

输入如下:



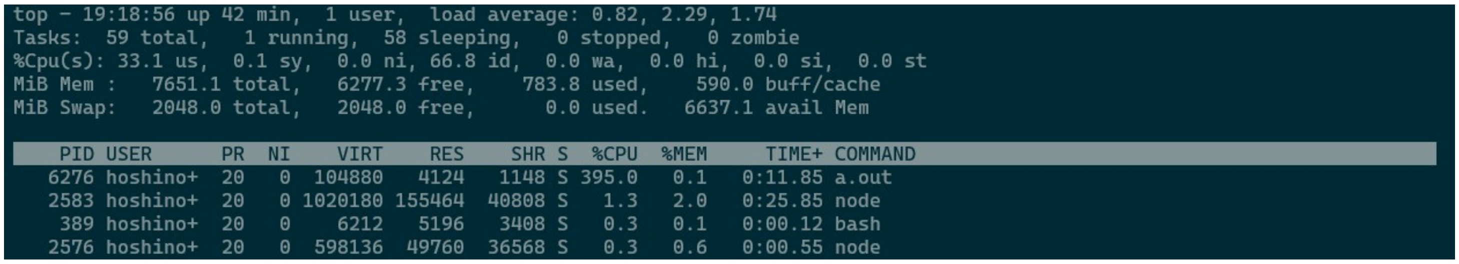
```
hoshino_july@LAPTOP-S40B7Q22:~/hpc_practice/lab5-thread/thread$ ls
common.h  common_threads.h  dgemm_chunk_multithread.c  t.c  t0.c  t0_without_thread_join.c  t1.c  thread_create_with_return_args.c  try.c
hoshino_july@LAPTOP-S40B7Q22:~/hpc_practice/lab5-thread/thread$ gcc dgemm_chunk_multithread.c -lpthread
hoshino_july@LAPTOP-S40B7Q22:~/hpc_practice/lab5-thread/thread$ ./a.out 400 200
```

运行结果如下:



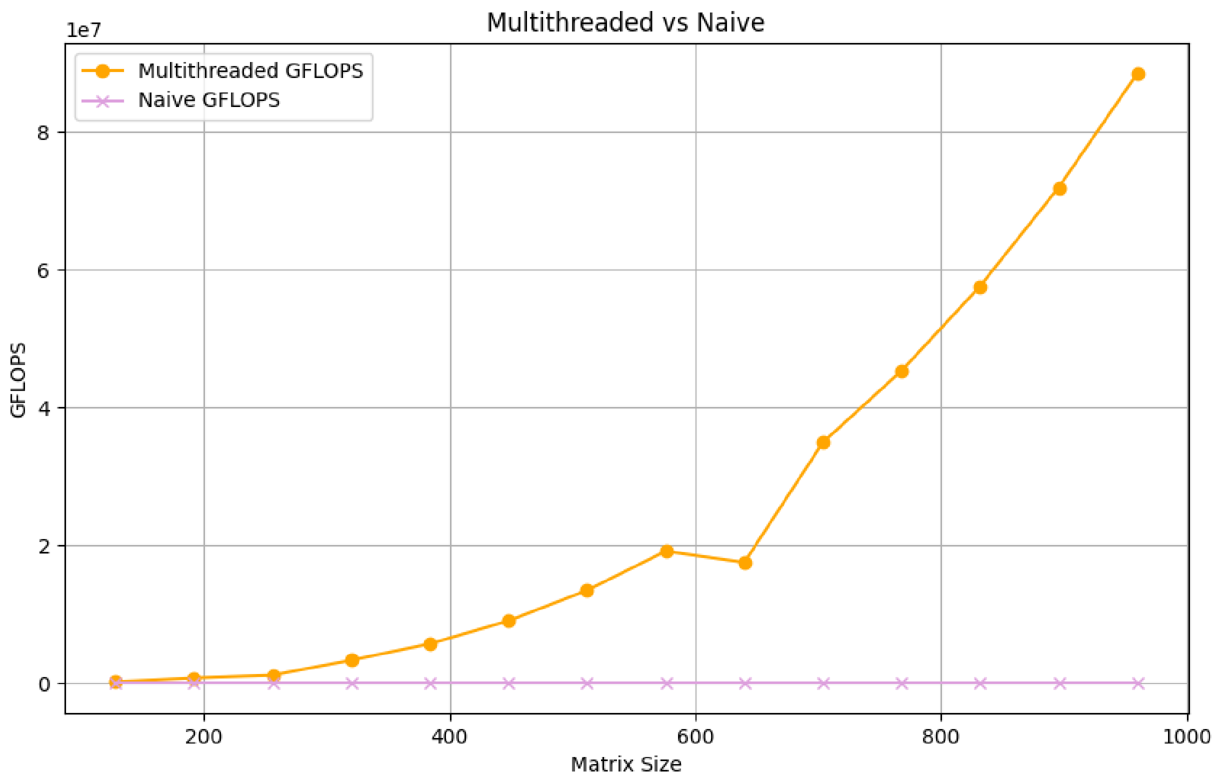
```
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     96
00.000000       9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.000000     9600.00000
0
Elapsed Time (s): 4.605142
GFLOPS: 0.042570
```

CPU利用率如下:



```
top - 19:18:56 up 42 min,  1 user,  load average: 0.82, 2.29, 1.74
Tasks:  59 total,   1 running,  58 sleeping,   0 stopped,   0 zombie
%Cpu(s): 33.1 us,  0.1 sy,  0.0 ni, 66.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7651.1 total,   6277.3 free,    783.8 used,    590.0 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   6637.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6276 hoshino+  20   0  104880   4124   1148 S 395.0   0.1   0:11.85 a.out
 2583 hoshino+  20   0 1020180 155464  40808 S   1.3   2.0   0:25.85 node
  389 hoshino+  20   0    6212   5196   3408 S   0.3   0.1   0:00.12 bash
 2576 hoshino+  20   0  598136  49760  36568 S   0.3   0.6   0:00.55 node
```

与Naive实现比较如下:



数据分析:

通过运行结果与以前的 `dgemm_naive` 代码性能相比可以发现,多线程分块极大地提高了矩阵乘法的性能。同时,根据cpu核数为12设置线程数量,也符合 `top` 查看到的CPU利用率变化。

碰到的问题及解决过程:

可能存在 **数据竞争** 问题,即由于线程不同步造成共享数据(如本代码中的矩阵C)被同时访问和修改而导致最终结果出错。
查询资料得知可以通过使用 **互斥锁** 等方法来确保只有一个线程可以修改共享数据。