lab6 实验报告

课程名 高性能计算应用实践
学期 2024年秋季学期
姓名 陈卫喆
学号 2023311F13

实验环境

OS版本

Ubuntu 22.04.3 LTS5.15.153.1-microsoft-standard-WSL2

gcc版本

gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

cpu型号

13th Gen Intel(R) Core(TM) i5-13500H

频率

cpu MHz : 3187.200

物理核数

Core(s) per socket: 8
Socket(s):1

内存大小

	total	used	free	shared	buff/cache	available
Mem:	8028508	833652	6938612	3268	256244	6958172
Swap:	2097152	0	2097152			

how-to-optimize-gemm各DGEMM实现及核心代码

naive_MMult

由$C(i,j) = \sum_{p=0}^{k-1} A(i,p) \times B(p,j)$
调用三层for循环得到结果，是最朴素的实现

```
int i, j, p;

for (i = 0; i < m ; i += 1)
{
    for (j = 0; j < n; j += 1)
    {
    for (p = 0; p < k; p += 1)
        {
            c[i * n + j] += a[i * k + p] * b[p * n + j];
        }
    }
}
```

openblas_MMult

直接调用cblas库实现

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
            m, n, k, 1.0, a, lda, b, ldb, 0.0, c, ldc);
```

multithread_MMult

将矩阵C分为多块，建立多线程，每个线程计算矩阵C的一部分

```
// 建立线程
for (m_start = 0, m_end = m_gap;
     m_start < m;
     m_start += m_gap, n_end += n_gap)
{
    for (n_start = 0, n_end = n_gap;
         n_start < n;
         n_start += n_gap, n_end += n_gap)
    {
        args_gemm_t args = {m_start, m_end, n_start, n_end, m, n, k, a, b, c};
        rc = pthread_create(&p[i++], NULL, naive_gemm, &args); assert (rc == 0);
    }
}

// 等待线程运行结束并回收资源
for (i = 0; i < CPU_CORES; i++)
{
    rc = pthread_join(p[i], NULL); assert (rc == 0);
}
```

openmp_MMult

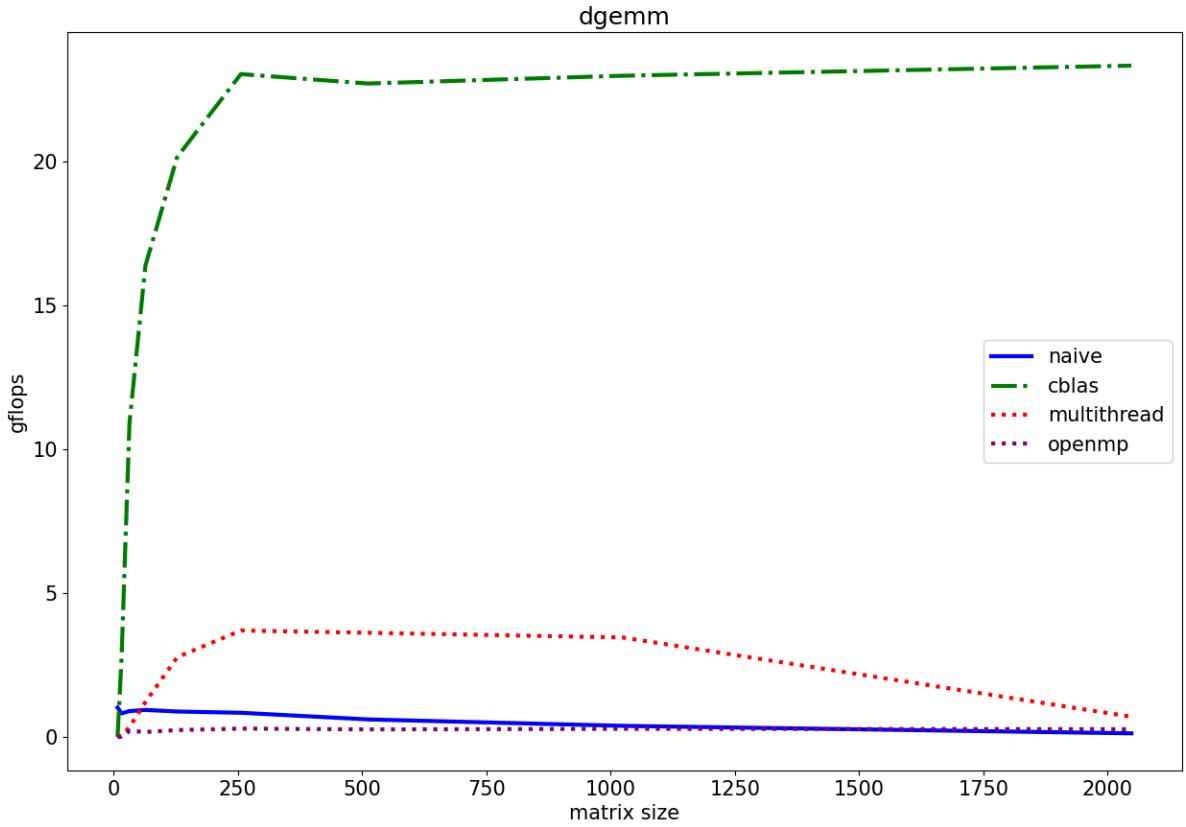
使用OpenMp Directives的parallel for命令
自动创建多个线程并将循环的工作分配给各个线程

```
int i, j, p;

#pragma omp parallel for
for (i = 0; i < m; i += 1)
{
    for (j = 0; j < n; j += 1)
    {
        for (p = 0; p < k; p += 1)
        {
            c[i * n + j] += a[i * k + p] * b[p * n + k];
        }
    }
}
```

gflops曲线图及汇总分析

gflops曲线图如下：



（矩阵规模为4096和8192时运行时间过长，很难出结果，所以我没有计算，希望老师理解）

从曲线图可以看出
大规模矩阵运算下，cblas的gflops远高于其他三种，其性能最优，除此之外，相比之下multithread优于剩下的两种。
但除cblas之外的三种方式在矩阵规模逐渐增大时gflops都会缩小并接近于一个很低的值，矩阵规模较大时性能也不佳。
综合来看cblas性能最好。

开启openmp时cpu利用率和多线程运行截图

cpu利用率：

```
top - 12:06:36 up  1:15,  1 user,  load average: 0.02, 0.16, 0.42
Tasks:  63 total,   2 running,  61 sleeping,   0 stopped,   0 zombie
%Cpu(s):  34.5/1.1    36[||||||||||||||||||||||||||||||||||||
MiB Mem :   7840.3 total,   6428.8 free,   1073.0 used,    338.5 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   6518.1 avail Mem
add filter #1 (ignoring case) as: [!]FLD?VAL
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  34651 yushik1   20   0  101444  39048   1968 R 554.0   0.5   0:12.02 test_MMult.x
    675 yushik1   20   0   21.4g 285900  53004 S   4.3   3.6   1:04.10 node
      1 root      20   0  165996  11308   8292 S   0.5   0.1   0:27.03 systemd
```

多线程：

```
                                                    └─10*[{node}]
├─SessionLeader───Relay(658)───node───6*[{node}]
├─SessionLeader───Relay(667)───node───6*[{node}]
├─SessionLeader───Relay(3041)───bash───make───sh───test_MMult.x───7*[{test_MMult.x}]
├─init───2*[{init}]
├─login───bash
└─{init-systemd(Ub}
```

lab3, lab5, lab6问题记录和回答

lab3

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？

从makefile文件的line 1, 2得到

```
OLD  := MMult0
NEW  := MMult1
```

OLD和NEW后面带的文件名即为可执行文件调用的函数，做对应更改即可
如将其改为

```
OLD  := openblas_MMult
NEW  := openblas_MMult
```

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。

makefile文件line 40 - 42:

```
@echo "date = '`date`';" > $(DATA_DIR)/output_$(NEW).m
@echo "version = '$(NEW)';" >> $(DATA_DIR)/output_$(NEW).m
$(BUILD_DIR)/test_MMult.x >> $(DATA_DIR)/output_$(NEW).m
```

执行了将结果写入_data目录下对应的.m文件的操作

lab5

cpu利用率：

```
top - 13:03:31 up  1:39,  1 user,  load average: 0.22, 0.11, 0.34
Tasks:  63 total,   2 running,  61 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.4 us,  0.1 sy,  0.0 ni, 93.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7840.3 total,   6429.9 free,   1051.4 used,    359.0 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   6538.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  44821 yushik1   20   0  101444  38932   1852 R 100.0   0.5   0:19.06 test_MMult.x
    675 yushik1   20   0   21.4g 286768  53004 S   1.0   3.6   1:23.86 node
      1 root      20   0  165996  11308   8292 S   0.7   0.1   0:34.71 systemd
```

多线程：

```
├─SessionLeader───Relay(3041)───bash───make───sh───test_MMult.x───7*[{test_MMult.x}]
├─SessionLeader───Relay(38860)───node───6*[{node}]
├─SessionLeader───Relay(38871)───node───6*[{node}]
├─init───2*[{init}]
```

lab6

1. 对变量shared, private属性及其可能引起的错误

parallel for会依据编译器自动分配线程并优化，因此不需多纠结，在最开始声明所有变量即可
为防止多线程可能引起的难以排查的问题，基本来说，应避免各线程访问同一个变量