# Lab6

## 实验环境

> OS: Ubuntu 22.04.3 LTS  5.15.153.1-microsoft-standard-WSL2 GNU/Linux
>
> gcc: version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)
>
> CPU: Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz  2 Cores 4 Threads

| | total | used | free | shared | buff/cache | available |
|---|---|---|---|---|---|---|
| Mem | 3974020 | 688848 | 2106220 | 3224 | 1178952 | 3040812 |
| Swap | 1048576 | | 1048576 | | | |

## 代码介绍

- naive_gemm：经典之作，代码结构清晰，除了跑得慢以外没有缺点。伟大，无需多言。

> 核心代码

```
 for (i = 0; i < m; i++) /* Loop over the rows of C */
  {
    for (j = 0; j < n; j++) /* Loop over the columns of C */
    {
      for (p = 0; p < k; p++)
      { /* Update C( i,j ) with the inner product of the ith row of A and
 the jth column of B */
        C(i, j) = C(i, j) + A(i, p) * B(p, j);
      }
    }
  }
```

- cblas：调用 OpenBlas 库，很好 blackbox，使我运算飞快。

> 核心代码

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

- pthread：调用 pthread 库，创建多线程并手动为每个线程分配任务。

> 核心代码

```
void MY_MMult(int m, int n, int k,
              double *a, int lda,
              double *b, int ldb,
              double *c, int ldc)
{
  pthread_t p[CPU_CORES];

  int i, rc;
  for(i=0;i<CPU_CORES;i++)
  {
    myarg_t args = {m, n, k,
                    a, lda,
                    b, ldb,
                    c, ldc,
                    i*m/CPU_CORES,
                    (i+1)*m/CPU_CORES};
    rc = pthread_create(&p[i], NULL, part, &args); assert(rc == 0);
  }
  for(i=0;i<CPU_CORES;i++)
  {
    rc = pthread_join(p[i], NULL); assert(rc == 0);
  }
}
```

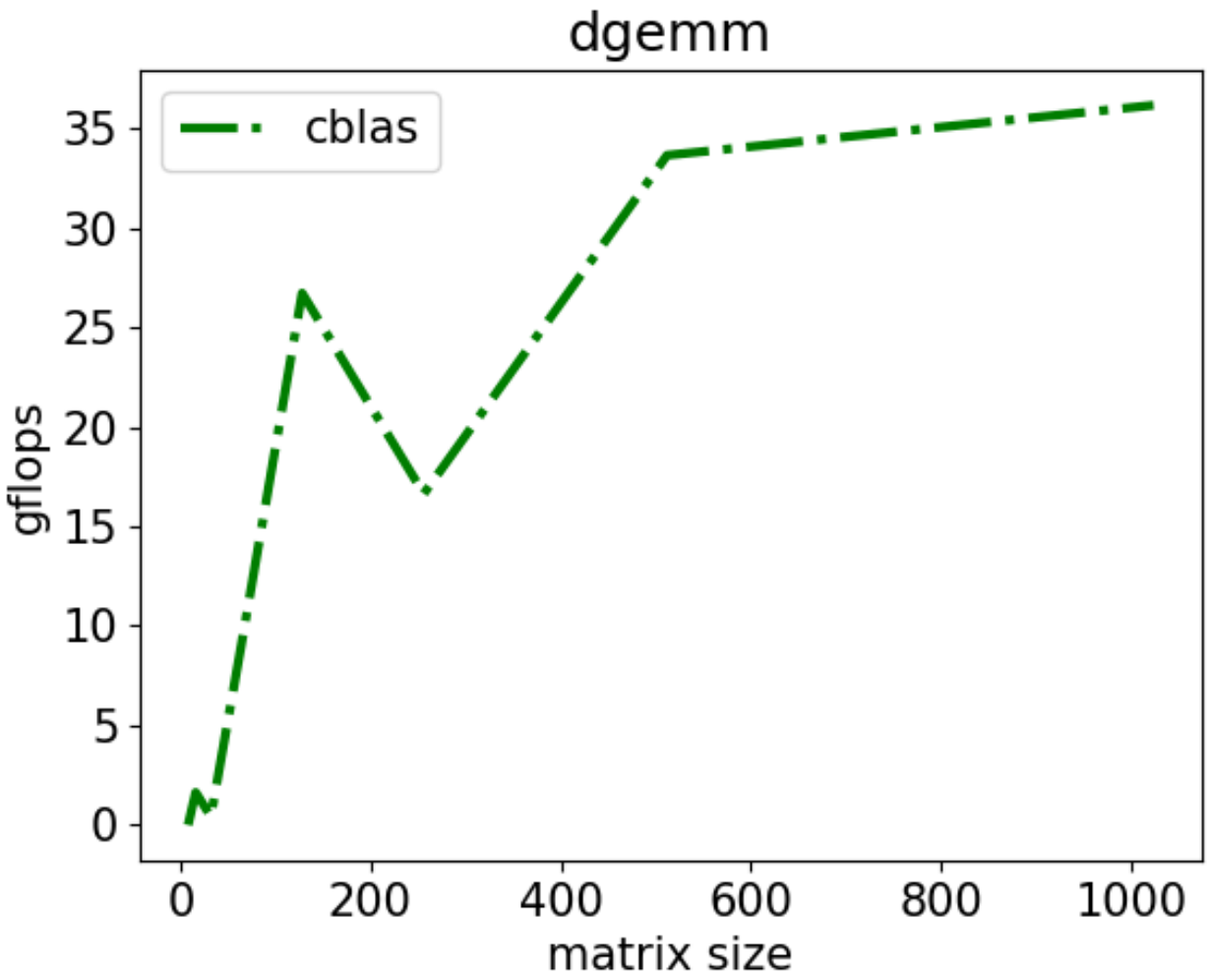
- openmp：使用 OpenMP API 自动创建并行化线程。

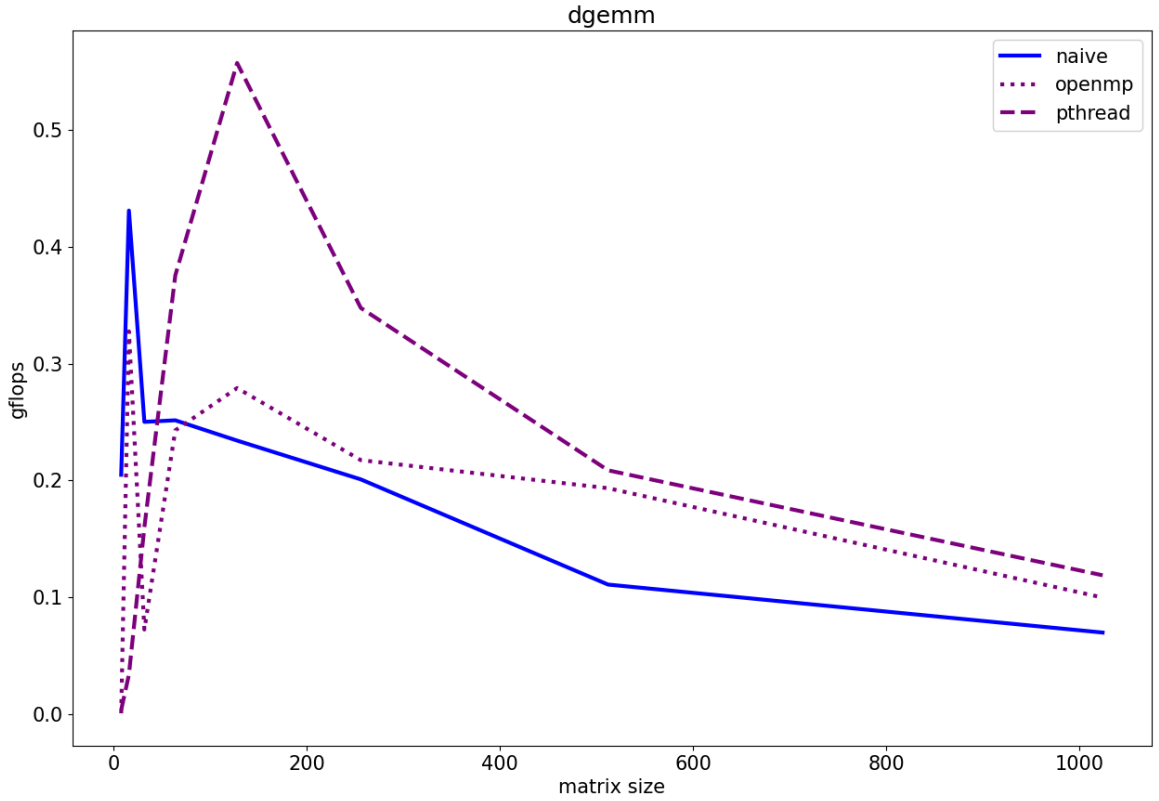
> 核心代码

```
void MY_MMult(int m, int n, int k,
              double *a, int lda,
              double *b, int ldb,
              double *c, int ldc)
{
  int i;
  #pragma omp parallel for
  for(i=0;i<CPU_CORES;i++)
  {
    myarg_t args = {m, n, k,
                    a, lda,
                    b, ldb,
                    c, ldc,
                    i*m/CPU_CORES,
                    (i+1)*m/CPU_CORES };
    part(&args);
```

```
17   }
18 }
```

**GFLOPS曲线图**






## 运行截图

```
top - 16:04:04 up 25 min,  1 user,  load average: 1.23, 0.75, 0.45
Tasks:  53 total,   2 running,  51 sleeping,   0 stopped,   0 zombie
%Cpu(s): 47.4 us,  1.2 sy,  0.0 ni, 44.3 id,  0.2 wa,  0.0 hi,  6.9 si,  0.0 st
MiB Mem :   3880.9 total,   2068.4 free,   1057.4 used,    755.1 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   2564.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  11823 hanami    20   0   68656  43160   1948 R 200.0   1.1   1:17.37 test_MMult.x
   9386 hanami    20   0   21.4g 250084  47852 S   2.7   6.3   0:26.78 node
      1 root      20   0  165984  11144   8124 S   0.7   0.3   0:19.16 systemd
   9419 hanami    20   0 1184200  66848  41040 S   0.7   1.7   0:03.35 node
    693 root      20   0   43388  36732  10104 S   0.3   0.9   0:12.87 python3
```

```
hanami@Hanami:~$ pstree
systemd─┬─2*[agetty]
        ├─cron
        ├─dbus-daemon
        ├─init-systemd(Ub─┬─SessionLeader───Relay(9307)───sh───sh───sh───node─┬─node───12*[{node}]
        │                 │                                                 ├─node─┬─cpptools───13*[{cpptools}]
        │                 │                                                 │      ├─node───6*[{node}]
        │                 │                                                 │      └─11*[{node}]
        │                 │                                                 ├─node─┬─bash───make───sh───test_MMult.x───3*[{test_MMult.x}]
        │                 │                                                 │      ├─bash───pstree
        │                 │                                                 │      ├─sh───cpuUsage.sh───sleep
        │                 │                                                 │      └─12*[{node}]
        │                 │                                                 └─10*[{node}]
```

## Lab3

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？

> makefile的链接规则末尾的目标文件会覆盖前面的，具体由以下代码决定

```
1 $(LINKER) $(OBJS) $(LDFLAGS) -o $@
```

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中包含echo指令，将性能数据写入文件

## Lab5截图