exp6

操作系统版本：Ubuntu 22.04.5 LTS

编译器版本：gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

cpu物理核数：24

cpu频率：最大2200MHz，最小800MHz

---

naive：朴素乘法，遍历C的每一个元素，将A的i行，B的j列的点乘作为Cij的值，总共的计算量是O(N^3)

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, j, p;

  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++)
      {
        C(i, j) = C(i, j) + A(i, p) * B(p, j);
      }
    }
  }
}
```

---

openblas：利用openblas库实现矩阵乘法加速

核心代码：

```
void cblasversion(int m, int n, int k, double *a, int lda,
              double *b, int ldb,
              double *c, int ldc){
  double alpha = 1, beta = 1;
  cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m, n, k, alpha, a, lda,
b, ldb, beta, c, ldc);
}
```

---

pthread：根据计算机的物理核数创建尽量多的线程，每个线程计算C的一部分实现加速

核心代码：

```
typedef struct mats
{
  double *a, *b, *c;
  int ord;
```

```
  int _M,_N,_K;
}mats;

void *mythread(void *arg)
{
  mats *mat = (mats *)arg;
  int ord = (mat->ord);
  int _M = mat->_M, _N = mat->_N, _K = mat->_K;
  for(int i=ord;i<_M*_N;i+=core_num)
  {
    for(int j=0;j<_K;j++)
    {
      mat->c[i]+=mat->a[(i/_N)*_K+j]*mat->b[j*_N+(i%_N)];
    }
  }
  return NULL;
}

void cstrct_mat(mats *S, double *a, double *b, double *c, int ord, int m, int n,
int k)
{
  S->a=a;
  S->b=b;
  S->c=c;
  S->ord=ord;
  S->_M = m;
  S->_N = n;
  S->_K = k;
}

void thread_gemm(int m, int n, int k, double *a, int lda,
              double *b, int ldb,
              double *c, int ldc)
{
  pthread_t P[core_num];
  int id;
  mats mat[core_num];

  for(int i=0;i<core_num;i++)
  {
    cstrct_mat(&mat[i], a, b, c, i, m, n, k);
    id = pthread_create(&P[i], NULL, mythread, &mat[i]);assert(id==0);
  }

  for(int i=0;i<core_num;i++)
  {
    id = pthread_join(P[i], NULL);assert(id==0);
  }
}
```

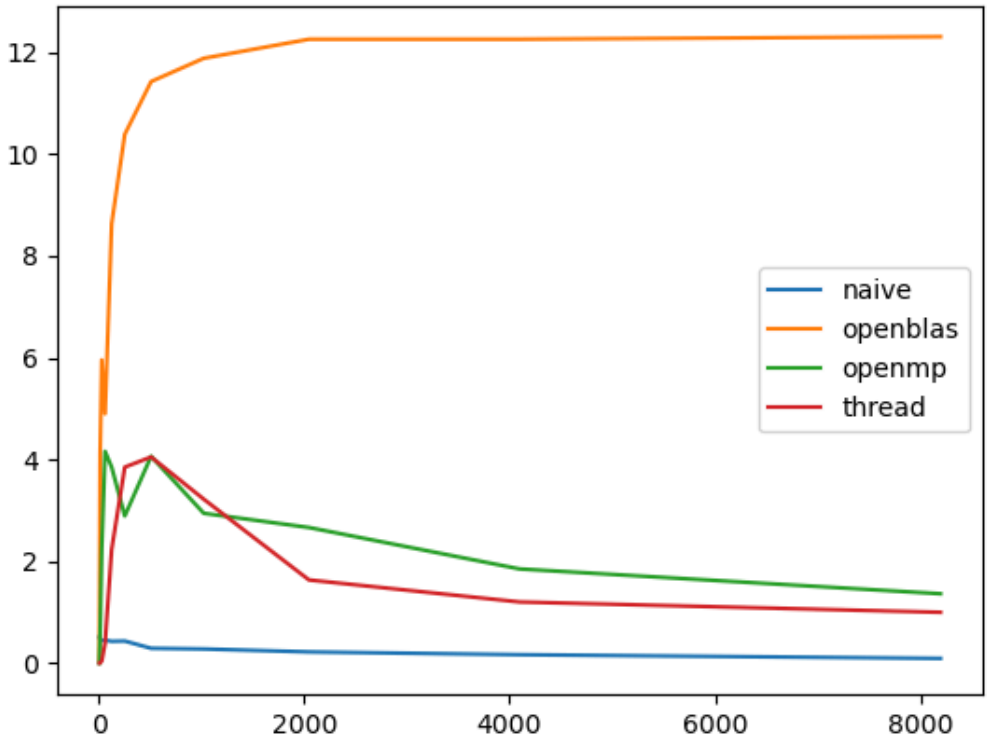
---

openmp：利用openmp的预编译指令优化矩阵乘法，同样是进行多线程并行计算

核心代码：

```
void OMP_MMult(int m, int n, int k, double *a, int lda,
               double *b, int ldb,
               double *c, int ldc)
{

#pragma omp parallel for
  for (int i = 0; i < m; i++) /* Loop over the rows of C */
  {
    for (int j = 0; j < n; j++) /* Loop over the columns of C */
    {
      for (int p = 0; p < k; p++)
      {
        C(i, j) = C(i, j) + A(i, p) * B(p, j);
      }
    }
  }
}
```



朴素乘法的gflops几乎不变都在10的-1次方数量级。自己编写的thread和openmp方法的gflops均随着矩阵规模的增大先增大后减少，最后稳定在1-2附近。openblas库方法的gflops几乎随着矩阵规模的增大一直变大，并很快稳定在12左右。

---

cpu利用率

```
进程号 USER      PR  NI    VIRT    RES    SHR    %CPU  %MEM     TIME+ COMMAND
 62101 thomas    20   0  866868 673612   3072 R  2394   4.2   7:17.08 test_MMult.x
 43825 thomas    20   0 1136.6g 434880 141280 S   7.9   2.7  11:15.94 code
  2837 thomas    20   0   10.9g 406260 147972 S   5.6   2.5  16:08.54 gnome-shell
 41813 thomas    20   0   32.6g 242212 227728 S   4.6   1.5   5:08.52 code
  2379 thomas    20   0   29.0g 437692 327544 S   4.0   2.7  22:31.04 Xorg
  2295 thomas    20   0 3599208  29436  21480 S   3.3   0.2  18:37.80 pulseaudio
   515 root      19  -1   94092  37856  35896 S   1.7   0.2   2:31.61 systemd-journal
  1523 root      20   0   99668  68948  12616 S   1.7   0.4   2:34.83 python3
   445 root     -51   0       0      0      0 S   0.7   0.0   2:06.85 irq/169-SYNA2BA6:00
   209 root      20   0       0      0      0 S   0.3   0.0   0:17.93 kauditd
  1313 root      20   0   14448   2080   1768 S   0.3   0.0   5:17.39 MvLogServer
  2142 nx        20   0 1518224  99560  14456 S   0.3   0.6   1:14.90 nxserver.bin
  4277 thomas    20   0 7186280 413300  51948 S   0.3   2.6   2:17.51 jetbrains-toolb
  4651 thomas    20   0 1124.0g 322012 222712 S   0.3   2.0   6:31.09 qq
 39586 thomas    20   0 1131.0g 137948  93300 S   0.3   0.9   0:32.76 qq
 41718 thomas    20   0 1132.2g 222508 151696 S   0.3   1.4   1:26.81 code
 44152 thomas    20   0 1133.9g 227548  64148 S   0.3   1.4   0:18.55 code
                   + CC util.c
```

多线程

```
        ├─code─┬─code───40*[{code}]
        │      └─code───41*[{code}]
        ├─code───10*[{code}]
        ├─code───25*[{code}]
        ├─code───15*[{code}]
        ├─2*[code───17*[{code}]]
        ├─code─┬─cpptools───41*[{cpptools}]
        │      └─15*[{code}]
        ├─code─┬─sh───cpuUsage.sh───sleep
        │      ├─2*[zsh]
        │      ├─zsh───make───sh───test_MMult.x───23*[{test_MMult.x}]
        │      └─18*[{code}]
        ├─code─┬─code───9*[{code}]
        │      ├─cpptools───41*[{cpptools}]
        │      ├─pet
        │      └─15*[{code}]
        └─36*[{code}]
    ├─3*[cpptools-srv───7*[{cpptools-srv}]]
    ├─dbus-daemon
    ├─dconf-service───2*[{dconf-service}]
    ├─2*[evince───5*[{evince}]]
    ├─evinced───2*[{evinced}]
    ├─evolution-addre───5*[{evolution-addre}]
    ├─evolution-calen───8*[{evolution-calen}]
```

---

lab3:

困难：

1.在添加cblas依赖的时候一开始不知道怎么做，后来尝试将-lcblas添加到LDFLAGS就可以编译了。

Q1:
OBJS := $(BUILD_DIR)/util.o $(BUILD_DIR)/REF_MMult.o $(BUILD_DIR)/test_MMult.o $(BUILD_DIR)/$(NEW).o $(BUILD_DIR)/cblas_MMult.o

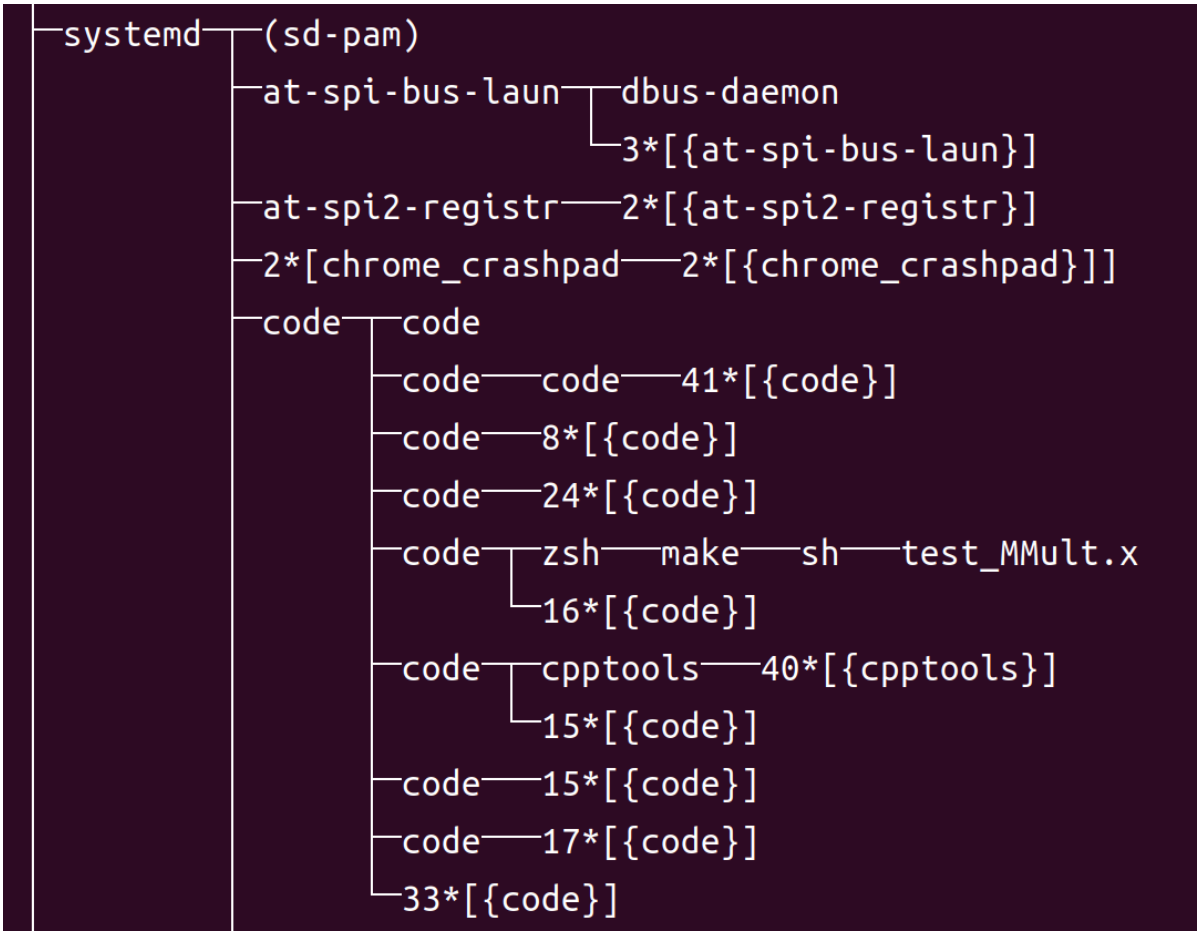
NEW变量决定了哪一个文件参与到编译过程中，也就决定我使用的是哪一个MY_MMult

Q2:

$(BUILD_DIR)/test_MMult.x >> $(DATA_DIR)/output_$(NEW).m

这句命令会将输出流重定向到输出文件中。

lab5:



```
top - 14:33:49 up 16 min,  1 user,  load average: 20.02, 7.55, 3.05
任务: 579 total,   2 running, 577 sleeping,   0 stopped,   0 zombie
%Cpu(s): 75.2 us,  0.6 sy,  2.6 ni, 21.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  15724.5 total,   4395.5 free,   4099.3 used,   7229.7 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   9833.6 avail Mem

  进程号 USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  28872 thomas    20   0  874756 672340   2560 S  2404   4.2  34:19.63 test_MM+
  34118 root      39  19  214080 189364 144640 S   3.0   1.2   0:02.70 apt-che+
    949 avahi     20   0    9648   4608   3328 S   2.0   0.0   0:14.41 avahi-d+
   2435 thomas    20   0   28.4g 240320 172940 S   1.3   1.5   0:32.21 Xorg
    242 root      20   0       0      0      0 I   1.0   0.0   0:00.37 kworker+
   2916 thomas    20   0   10.3g 345076 149888 S   1.0   2.1   0:26.99 gnome-s+
  32826 thomas    20   0 1123.7g 281088 187628 S   1.0   1.7   0:12.87 qq
   4982 thomas    20   0  980532  66196  51076 S   0.7   0.4   0:02.28 gnome-t+
  31377 thomas    20   0   16720   4096   3072 R   0.7   0.0   0:00.45 top
    441 root     -51   0       0      0      0 S   0.3   0.0   0:03.23 irq/169+
    618 root      20   0       0      0      0 I   0.3   0.0   0:00.99 kworker+
    620 root      20   0       0      0      0 I   0.3   0.0   0:00.64 kworker+
   1084 root      20   0   14448   1536   1536 S   0.3   0.0   0:09.64 MvLogSe+
   5272 thomas    20   0 1131.9g  96040  71936 S   0.3   0.6   0:01.59 code
   7795 thomas    20   0 1608448  19400  12544 S   0.3   0.1   0:00.04 snap
  33774 thomas    20   0 1130.9g 127780  87552 S   0.3   0.8   0:00.43 qq
      1 root      20   0  168328  11264   7168 S   0.0   0.1   0:01.85 systemd
```

问题：编写多线程程序的时候传递参数没有用地址而是用值，这导致所有线程计算的都是同一个位置上的值，将传递的结构体参数改成传递地址之后问题解决了。