# gcc 使用和矩阵乘法性能比较

**信息**

| 姓名 | 学号 | 编辑日期 |
| --- | --- | --- |
| 梁鑫嵘 | 200110619 | @September 18, 2021 |



## 实验简介

### gcc

GCC（GNU Compiler Collection，GNU编译器套件）是由GNU开发的编程语言译器。本试验学习了这个编译器的使用。

### OpenBLAS

OpenBLAS 是一个开源的矩阵计算库，包含了诸多的精度和形式的矩阵计算算法。本实验实现了在 Linux gcc 编译器下对这个矩阵库的使用，以及实现了这个矩阵库的矩阵乘法的运行效率和普通矩阵乘法的比较。

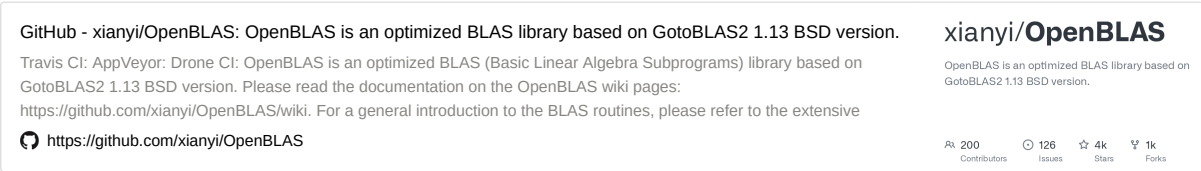
## Linux 下 gcc 的使用

当我们编译生成一个可执行文件时，我们一般直接执行：`gcc -o test test.c` 。这个命令将直接在当前目录生成一个 `test` 可执行文件，不留下任何其他文件。 `./test` 即可执行这个文件。

但是，这个编译过程的中间细节是什么样呢？我们可以手动通过改变编译参数的方式改变 GCC 的编译过程，使之生成一系列中间文件。命令如下：

1. `gcc -E test.c -i -o test.i` ：生成预处理文件
2. `gcc -S test.i -o test.S` ：根据预处理文件生成汇编码
3. `gcc -c test.S -o test` ：根据汇编码汇编成链接文件
4. `gcc test.o -o test` ：把链接文件链接成为可执行文件

### OpenBLAS 的编译和使用




OpenBLAS 开源于：https://github.com/xianyi/OpenBLAS 。

编译方式很简单，在源码根目录执行 `make` 即可编译生成二进制链接库， `make install PREFIX=<安装目录>` 即可安装文件到指定目录。我这里安装到 `/usr/share/openblas` 。

但是 `make install` 之后，GCC可能仍然找不到文件。我们需要把这些文件再添加到系统的目录中。

```
# 软链接到系统的 lib 目录
ln -s /usr/share/openblas/lib/libopenblas.so /usr/local/lib/libopenblas.so
```

```
# 让 CMake 找到依赖文件
ln -s /usr/share/openblas/lib/cmake/openblas openblas
```

包含目录的话暂时不添加，在编译自己程序的命令行里再添加。

使用 OpenBLAS 只需要 `#include "cblas.h"` ，然后 `cblas_dgemm` 的定义就在这个头文件里。

```
void cblas_dgemm(OPENBLAS_CONST enum CBLAS_ORDER Order, OPENBLAS_CONST enum CBLAS_TRANSPOSE TransA, OPENBLAS_CONST enum CBLAS_TRANSPOS
     OPENBLAS_CONST double alpha, OPENBLAS_CONST double *A, OPENBLAS_CONST blasint lda, OPENBLAS_CONST double *B, OPENBLAS_CONST blasi
```

代码示例：

```
  const enum CBLAS_ORDER Order = CblasRowMajor;
  const enum CBLAS_TRANSPOSE TransA = CblasNoTrans;
  const enum CBLAS_TRANSPOSE TransB = CblasNoTrans;
  int lda = K;  // A的列
  int ldb = N;  // B的列
  int ldc = N;  // C的列
  // C = alpha * AB + beta * C
  cblas_dgemm(Order, TransA, TransB, M, N, K, 1, A->content, lda, B->content,
              ldb, 0, C->content, ldc);
```

编译命令行：

```
gcc ${build_path}/mat.o \
    ${build_path}/mat_mul_test.o \
    -o ${build_path}/mat_mul_test \
    -lm \
    -L${OpenBLAS_LIBRARIES} \
    -lopenblas \
    -static \
    -lpthread
```

由此写出整个工程的 Makefile：

```
build_path:=build_mkfile
OpenBLAS_INCLUDE_DIRS:=/usr/share/openblas/include
OpenBLAS_LIBRARIES:=/usr/share/openblas/lib

build:
# 神创造了世界，神的福音润泽世间。
  $(shell if [ ! -d ${build_path} ]; then mkdir ${build_path}; fi)

# 首先是犯下懒惰之罪的 mat.c，
# 仗着自己有一些矩阵运算函数，
# 就希望别的文件来静态链接它。
  gcc -E src/mat.c -o ${build_path}/mat.i
  gcc -S ${build_path}/mat.i -o ${build_path}/mat.s
  gcc -c ${build_path}/mat.s -o ${build_path}/mat.o
# 然后是犯下傲慢之罪的 mat_mul_test.c，
# 自己只实现了一个函数就敢教贵为 OpenBLAS 的 cblas_dgemm 做事，
# 没有任何一点自知之明。
  gcc -E src/mat_mul_test.c -o ${build_path}/mat_mul_test.i -I${OpenBLAS_INCLUDE_DIRS}
  gcc -S ${build_path}/mat_mul_test.i -o ${build_path}/mat_mul_test.s
  gcc -c ${build_path}/mat_mul_test.s -o ${build_path}/mat_mul_test.o
# 最后是犯下暴食之罪的 mat_mul_test，
# 它疯狂吞噬了 mat.o 和 mat_mul_test.o 不算，
# 还残忍吞入了 libopenblas.so，实在是欲望滔天！
  gcc ${build_path}/mat.o \
    ${build_path}/mat_mul_test.o \
    -o ${build_path}/mat_mul_test \
    -lm \
    -L${OpenBLAS_LIBRARIES} \
    -lopenblas \
    -static \
    -lpthread

run: build
# 所以，神降下了祂的神罚，惩罚作为世界载体的 CPU 高负荷执行一段无用的计算
# 然后将此记录于石碑之上。
  ./build_mkfile/mat_mul_test 256

clean:
# 仁慈的父，我已坠入，
# 看不见罪的国度。
# 请原谅我，我的自负，
```

```
# 盖着一道孤独。
  rm -rf build_mkfile/
```

使用 `make` 即可编译， `make run` 即可运行。

当然，我们也可以用 CMake 进行编译。由于我们已经将对应的 `.config` 添加到系统中，我们可以用如下代码添加依赖和链接库：

```
find_package(OpenBLAS REQUIRED)
include_directories(${OpenBLAS_INCLUDE_DIRS})
message(STATUS "OpenBLAS include path: ${OpenBLAS_INCLUDE_DIRS}")

link_libraries(-lpthread ${OpenBLAS_LIBRARIES} -lm)
```

（完整 `CMakeLists.txt` 见代码中。）

使用 CMake 编译：

```
mkdir build
cd build
cmake ..
make
```

## 效率比较

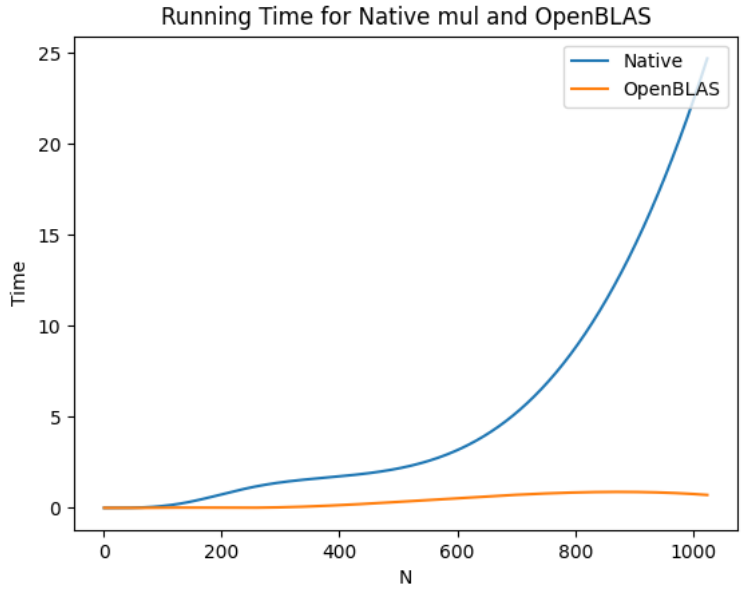
本实验的编译、运行、比较数据代码都写到 `mul_test_all.py` 文件当中了。查看这个文件的使用帮助： `python3 mul_test_all.py -h`

```
➜  lesson02 git:(master) ✗ python3 mul_test_all.py -h
usage: mul_test_all.py [-h] [-l] [-m MAX_N] [-b {make,cmake}]

optional arguments:
  -h, --help            show this help message and exit
  -l, --show-log        绘制 log 图像
  -m MAX_N, --max-n MAX_N
                        设置最大 N，N = 2^k，请输入 k。
  -b {make,cmake}, --build-method {make,cmake}
                        设置编译方式，两者都可以。
➜  lesson02 git:(master) ✗
```

我们运行：`python3 mul_test_all.py -m 10 -b make` ，将会在 `data/` 得到一张图片，即为本实验的比较数据结果。

```
➜  lesson02 git:(master) ✗ python3 mul_test_all.py -m 10 -b make
Building excutable file by Make: mat_mul_test...
gcc -E src/mat.c -o build_mkfile/mat.i
gcc -S build_mkfile/mat.i -o build_mkfile/mat.s
gcc -c build_mkfile/mat.s -o build_mkfile/mat.o
gcc -E src/mat_mul_test.c -o build_mkfile/mat_mul_test.i -I/usr/share/openblas/include
gcc -S build_mkfile/mat_mul_test.i -o build_mkfile/mat_mul_test.s
gcc -c build_mkfile/mat_mul_test.s -o build_mkfile/mat_mul_test.o
gcc build_mkfile/mat.o build_mkfile/mat_mul_test.o -o build_mkfile/mat_mul_test -lm -L/usr/share/openblas/lib -lopenblas -static -lpth
N =    1, time_native =   1.4e-05, time_OpenBLAS =  0.002502
N =    2, time_native =   2.2e-05, time_OpenBLAS =    0.0002
N =    4, time_native =   2.5e-05, time_OpenBLAS =   0.00021
N =    8, time_native =   5.9e-05, time_OpenBLAS =  0.000191
N =   16, time_native =  0.000365, time_OpenBLAS =  0.000224
N =   32, time_native =  0.002969, time_OpenBLAS =  0.000376
N =   64, time_native =  0.022499, time_OpenBLAS =  0.000921
N =  128, time_native =  0.224119, time_OpenBLAS =  0.016948
N =  256, time_native =  1.163516, time_OpenBLAS =  0.010383
N =  512, time_native =  2.248667, time_OpenBLAS =  0.355198
N = 1024, time_native = 24.685139, time_OpenBLAS =  0.712758
Saved image file: ../data/plot.png
```




## 比较结论

OpenBLAS 对普通的矩阵运算进行了优化，在我的理解中，优化点主要有：

1. 优化了 Cache 命中率，把高频用到的数据放到 CPU 的 Cache 中，减少因为多次读写内存造成的延迟。
2. 对目标设备进行优化，充分利用设备性能，比如使用更加适合平台的指令集，或者充分发挥处理器多发射优势，进行循环展开等。
3. 对数据进行分块，把数据部分拷贝到连续空间内，减小访问距离。

以下是作者的 OpenBLAS 库的思路。

https://www.leiphone.com/category/yanxishe/Puevv3ZWxn0heoEv.html

可以清楚看到，OpenBLAS 对于简单的矩阵运算效率提升明显。当 N = 1024 时，效率甚至是简单算法的约 37 倍。通过各种针对算法和硬件的优化，矩阵运算能够被优化到很快很快。

希望在不久以后我也能拥有这样神奇的“魔法”，让算法在硬件上发挥出最好性能。