# 实验二：Linux环境下C语言编程

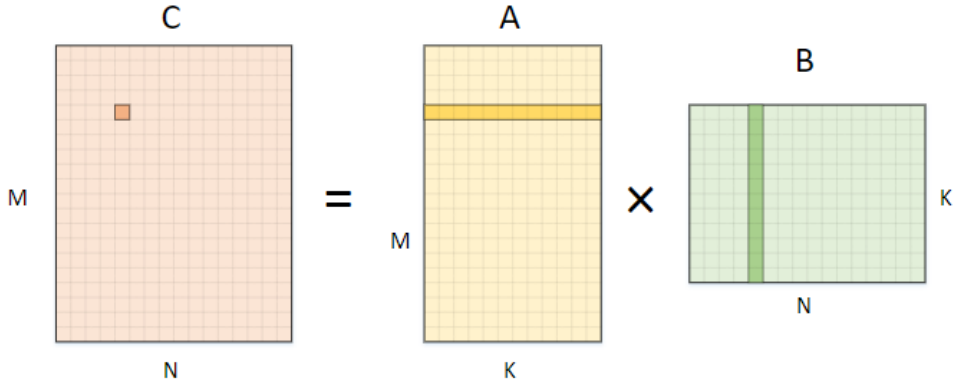
郑海刚

# 本讲概述

- 主要内容
  - naive gemm
  - 可执行文件生成
  - blas & openblas
  - 时间测量

# 通用矩阵乘法：GEMM

- GEMM (General matrix multiply)
  - 一个被广泛使用的基础算法，例如深度学习中的卷积操作
  - 双精度浮点简写为DGEMM，单精度浮点简写SGEMM



# naive dgemm（朴素版本）

- lab2/dgemm_naive.c
- gcc dgemm_naive.c编译
- ./a.out执行
  - 只输入a.out是没法执行的

```
void dgemm(int m, int n, int k, int beta,
           double A[][k], double B[][n], double C[][n]){
    for(int i=0; i< m;i ++){    //C[i]
        for(int j=0; j< n; j++){  //C[i][j]
            C[i][j] = beta*C[i][j];
            for(int p=0; p< k; p++){
                C[i][j] += A[i][p]*B[p][j];
            }
        }
    }
}
```

- gcc dgemm_naive.c && ./a.out  编译后直接执行

# 可执行文件生成：编译、汇编、链接

- gcc xx.c 生成a.out可执行文件， 经历了什么过程?
  - 预处理：gcc –E xx.c -o xx.i 生成.i文件
  - 编译：gcc –S xx.i -o xx.s生成汇编代码文件
  - 汇编：gcc –c xx.s -o xx.o生成可重定位目标文件
  - 链接：gcc xx.o xx.o 合并生成最后的可执行文件
    - 静态链接
    - 动态链接

# 链接：lab2/link

- 将多个可重定位对象文件拼接在一起
  - 多个C代码项目，存在互相调用
- 只有一个文件是否要链接?
  - 也需要，调用库函数如printf需链接标准库
  - 即使只有main函数，也需要链接到运行时库

```
$ lab2/link »ls
main.c  sum.c
$ lab2/link »cat main.c
#include<stdio.h>

int sum(int n);
int main(){
    printf("%d\n", sum(100));
}%
$ lab2/link »cat sum.c
int sum(int n){
    int sum=0;
    for(int i=1; i<=n; i++){
        sum += i;
    }
    return sum;
}%
$ lab2/link »gcc -c main.c sum.c
$ lab2/link »ls
main.c  main.o  sum.c  sum.o
$ lab2/link »gcc main.o sum.o
$ lab2/link »ls
a.out   main.c  main.o  sum.c  sum.o
```

# 编译、链接

- 难点是编译、链接
  - 预处理主要复制粘贴、去除注释，相对比较简单
  - 汇编：汇编代码到机器码，基本是一对一映射
- 为什么要了解这样的过程?
  - 作为计算机专业的同学需要探知原理
  - 大型项目的构建需要，知道编译链接的过程才知道怎么写makefile
  - 编译出错的时候，更容易解决问题

# BLAS（基础线性代数程序集）

- BLAS(Basic Linear Algebra Subprograms)
  - 是一套标准，包含了低级别的线性代数操作
    - level1：向量操作
    - level2：矩阵向量操作
    - level3：矩阵矩阵操作
  - http://www.netlib.org/blas/ 提供了API定义同时提供了参考实现BLAS
  - 有多种实现：**GotoBLAS**、 cuBLAS、Intel MKL、

# OpenBlas介绍

- 基于GotoBLAS2 1.13 BSD version实现的高性能BLAS库
- OpenBlas怎么用
  - RTFM： https://github.com/xianyi/OpenBLAS/wiki/Document
  - 源码编译安装： 可以安装最新的版本， 只需make一下
  - 预编译：sudo apt install libopenblas-dev
  - 预编译默认装在系统路径
    - /usr/lib/x86_64-linux-gnu/
    - /usr/include/x86_64-linux-gnu/

# User Manual: OpenBlas Code examples（1）

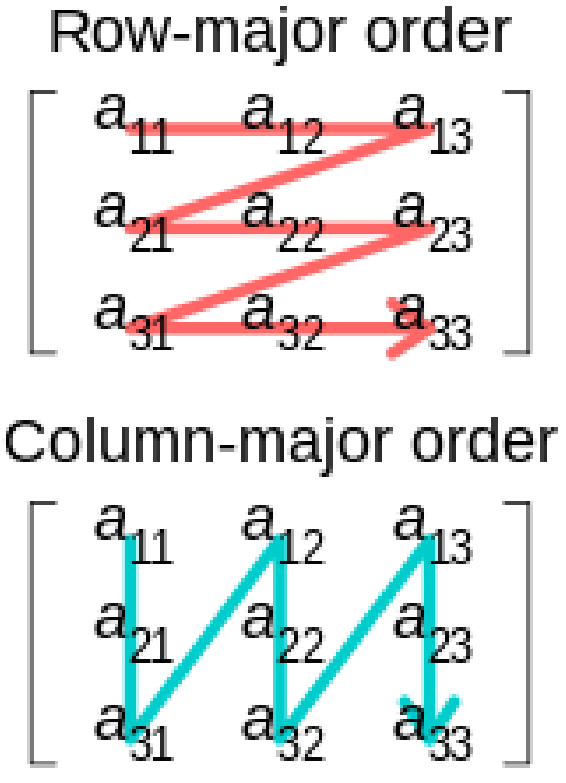
- lab2/test_cblas_dgemm.c：
  - 编译：gcc -o test_cblas_open test_cblas_dgemm.c -lopenblas

# cblas_dgemm()接口参数

- MKL接口说明：　C := alpha*op(A)*op(B) + beta*C
- 参数说明
  - layout：行主序或列主序
  - transa, transb：　矩阵A、B是否转置或共轭转置
  - m, n, k：实际计算的矩阵的行列数
  - alpha, beta：　系数
  - a, b, c ：指向原始矩阵A、B、C的指针，一维数组
  - lda, ldb, ldc ：a,b,c指向的矩阵的 leading dimension

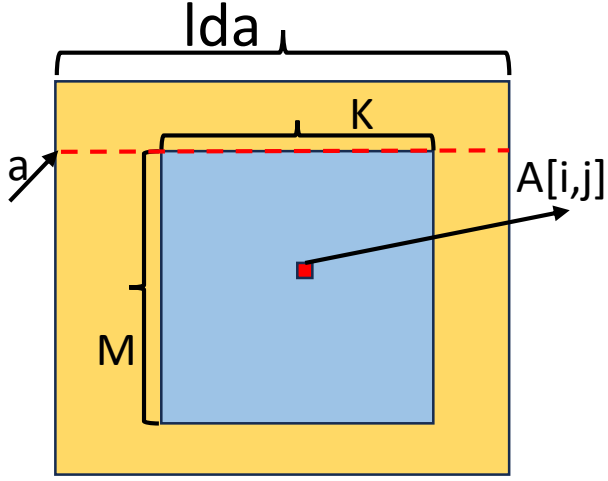
# 行主序、列主序（也叫行优先、列优先）

- Row- and column-major order Wikipedia
  - 多维数组在线性的存储空间中的存储方式
  - 行主序：行的连续元素相邻
  - 列主序：列的连续元素相邻
- lab2/row_major.c
  - 打印二维数组的元素地址
  - C语言二维数组是连续存储，行主序
  - Fortran是列主序
- CPU缓存，访问相邻地址的数据会更快



# lda、ldb、ldc

- leading dimension： 如果是行主序，就是每行的元素个数；列主序是每列的元素个数。
- 有了m,n,k三个参数为什么还需要lda,ldb,ldc?
  - 大矩阵分块计算时只用到一部分数据
  - 行主序无转置的情况：lda >= k




lda

Specifies the leading dimension of *a* as declared in the calling (sub)program.

| | transa=CblasNoTrans | transa=CblasTrans or transa=CblasConjTrans |
|---|---|---|
| Layout = CblasColMajor | lda must be at least max(1, m). | lda must be at least max(1, k) |
| Layout = CblasRowMajor | lda must be at least max(1, k) | lda must be at least max(1, m). |

# OpenBlas Code examples（2）

- lab2/time_dgemm.c
  - 编译：gcc -o time_dgemm time_dgemm.c –lopenblas
  - 运行：./time_dgemm 1024
  - 除了printf输出，还生成了timeDGEMM.txt文件
  - 数组空间由malloc分配，固定初始化或随机初始化
  - 时间测量： gettimeofday
  - gflops：浮点操作次数（4*m*n*k）除以时间

# 时间测量：time命令

- time ls

```
:~/hpc/tmp$ time ls

real    0m0.002s
user    0m0.001s
sys     0m0.001s
```

- What do 'real', 'user' and 'sys' mean in the output of time(1)?
  - real：时钟时间（墙上时间），有些shell下是total
  - user：用户cpu时间
  - sys：系统cpu时间
  - cpu时间=user+sys
  - (user + sys)/real就是程序的cpu利用率，要考虑多个CPU的情况

# CPU时间和时钟时间

- 无限循环：lab2/while.c
  - 编译：gcc while.c
  - 运行：time ./a.out ，使用ctrl+c终止
  - real time > cpu time (user+sys)
    - sleep把进程切换出去
    - sleep的时间不计入cpu但计入real
  - 无限循环cpu还能执行其他程序
    - 时钟中断，操作系统会把进程切换出去

```c
#include<stdio.h>
#include <unistd.h>

int main(){
    int a = 0;

    while(1){
        a++;
        if( (a%100000000) == 0){
            sleep(1);
            printf("a = %d\n", a);
        }
    }
}
```

```
:~$ time ./a.out
a = 100000000
a = 200000000
a = 300000000
a = 400000000
a = 500000000
^C
real    0m6.558s
user    0m1.103s
sys     0m0.000s
```

# 时间测量：代码中使用测量函数

- time命令只能粗略测整个程序的运行时间，精度是ms级
- gettimeofday函数，返回时间戳，精度是us级
  - Unix time 时间戳， https://www.unixtimestamp.com/在线转换
  - man gettimeofday
    - struct timeval *tv

```
struct timeval {
    time_t      tv_sec;     /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

# Git仓库登记

- https://docs.qq.com/sheet/DWmx4R2tZdWFiYmNp?tab=BB08J2