中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

并行程序设计与算法（实验）

Lab0-环境配置与串行矩阵乘法

吴迪

中山大学 计算机学院

实验概要

- 内容
  - 配置环境
  - 实现串行矩阵乘法
  - 优化串行矩阵乘法
  - 对比及初步性能分析

- 目标
  - 熟悉Linux编程及编译环境（shell, vim, gcc, gdb）
  - 掌握基本矩阵乘法实现方式

环境配置

- 1. 下载安装虚拟机软件（推荐VMware或VirtualBox）
  - VMware: https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html
  - VirtualBox：https://www.virtualbox.org/wiki/Downloads
- 2. 下载Ubuntu 18.04镜像文件：
  - http://releases.ubuntu.com/18.04/

软件安装

- 安装Ubuntu
  - 创建虚拟机实例，设置CPU核、内存、硬盘容量
  - 在虚拟光驱中加载Ubuntu镜像文件（.iso），并安装操作系统

- 安装OpenMPI 和 vim
  - 启动系统，在命令行终端安装OpenMPI
    - `sudo apt-get update`
    - `sudo apt-get install libopenmpi-dev -y`
    - `sudo apt-get install vim -y`

Intel MKL库介绍

- Intel Math Kernel Library（MKL）是英特尔公司推出的一款高性能数学库
  - 旨在提供数值计算、线性代数、信号处理和统计分析等方面的优化函数和子程序
  - MKL库针对英特尔处理器架构进行了高度优化，可以充分发挥英特尔处理器的性能优势
  - 除了数学函数之外，Intel MKL库还提供了一些优化工具，例如性能分析器和线程构建器

软件安装

- Intel oneAPI Math Kernel Library (MKL) 命令行安装

```
// 下载intel 公钥
sudo wget https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
sudo apt-key add GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB

// Add the APT Repository
sudo sh -c 'echo deb https://apt.repos.intel.com/mkl all main > /etc/apt/sources.list.d/intel-mkl.list'
sudo apt-get update

// Install
sudo apt-get install intel-mkl-64bit-2020.2
source /opt/intel/compilers_and_libraries_2020/linux/mkl/bin/mklvars.sh intel64 ilp64
```

Intel MKL gcc 编译运行示例

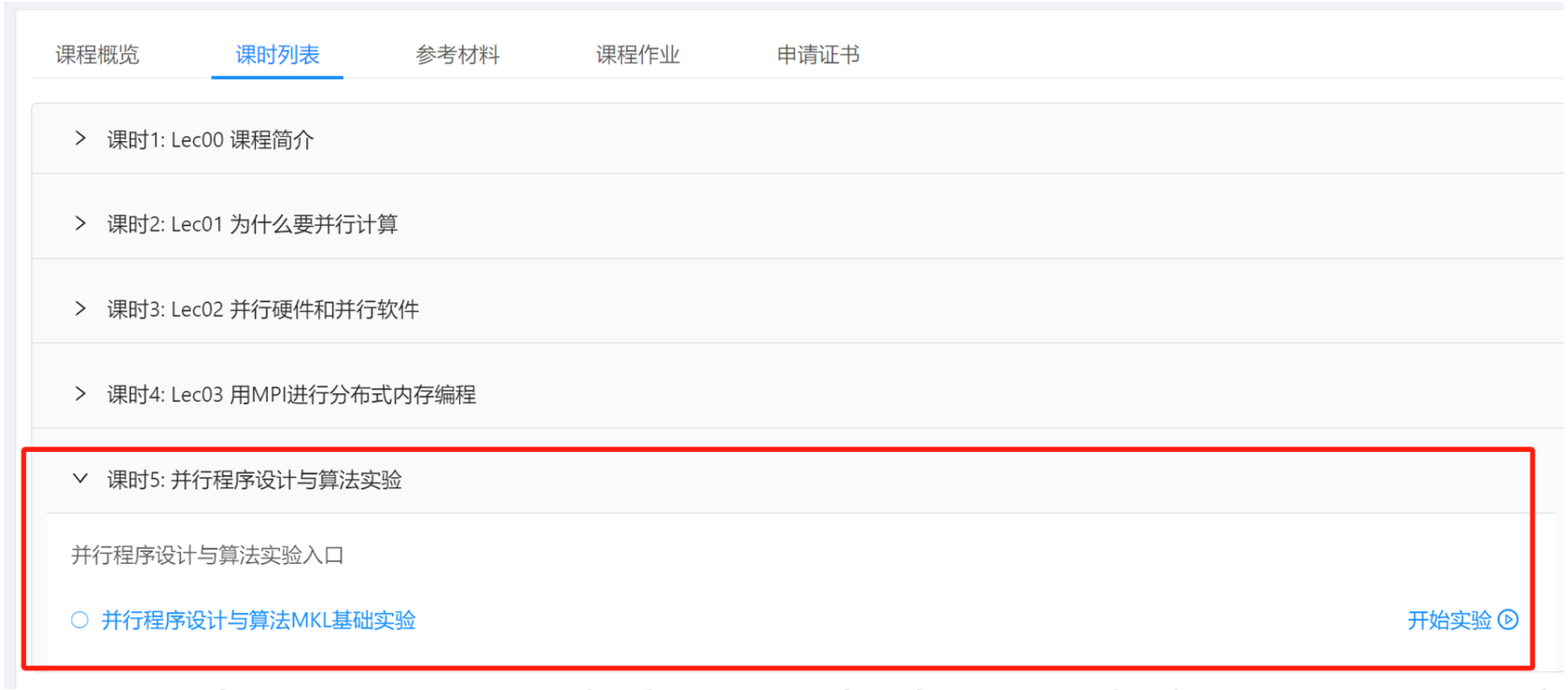
```
. /opt/intel/bin/compilervars.sh intel64

gcc example.c -lmkl_rt

./a.out
```

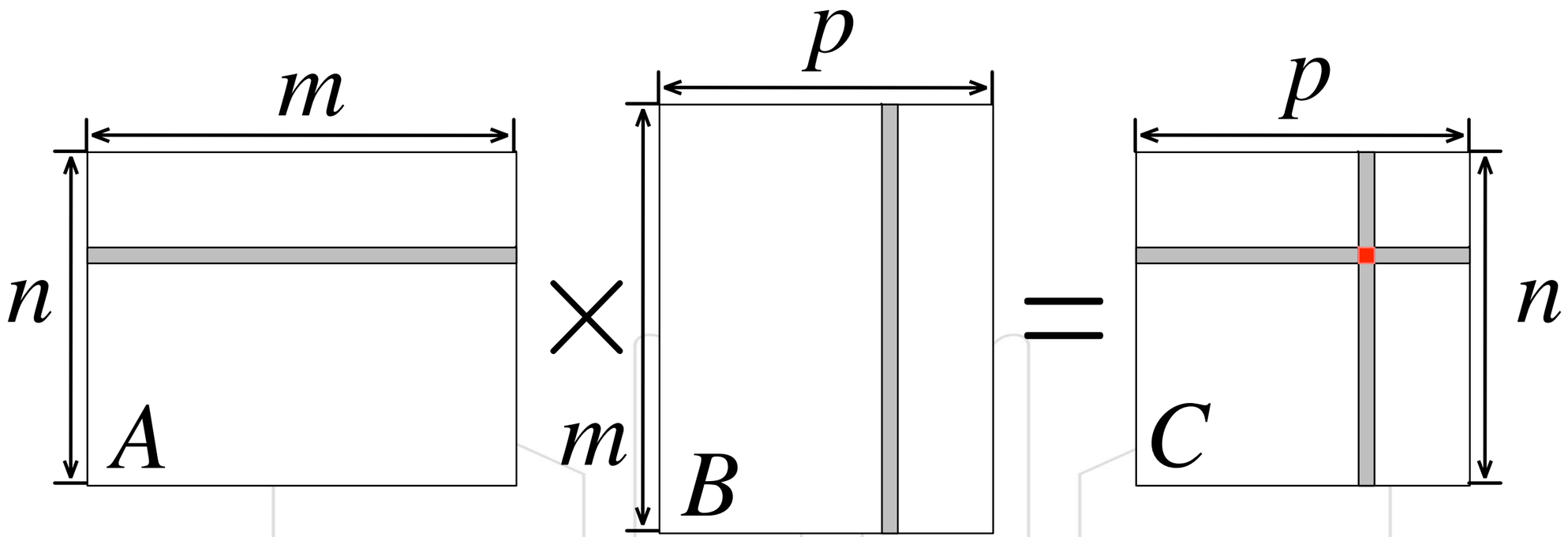
超算习堂环境

- 在“课时5：并行程序设计与算法实验”中，提供了预装MKL和openmpi的基础实验环境
  - MKL配置指引（user-guide.txt）和示例代码（example.c）
  - 用户数据可以保存在目录 ~/data/中



矩阵乘法

- 对于输入矩阵A, B计算其乘积$C = A \cdot B$
  - 对于C中每个元素，计算$C_{ij} = \sum_{k=1}^{m} A_{ik} \cdot B_{kj}$
    - 共需$n \times p$次向量内积
    - 每次内积需m次乘法及$m - 1$次加法



串行矩阵乘法优化

- 类别1：实现优化 （从计算机体系结构角度优化）
  - 调整编译器选项
    - 设置优化级别、编译器标志等
  - 调整循环顺序
  - 循环展开等
  - 参考资料
    - https://jackwish.net/2019/gemm-optimization.html
    - https://blog.csdn.net/weixin_43614211/article/details/122105195
- 类别2：算法优化 （从数学角度优化）
  - Strassen算法，Coppersmith-Winograd算法

GCC编译优化

编译优化级别

- -O0（不优化）：默认的编译选项，表示不进行任何优化。
- -O（优化）：这个选项会开启一些基本优化，是 -O1 的一个简化版本，通常用于快速测试编译器是否能够生成有效的代码。
- -O1：这个级别会开启一系列的优化，包括循环展开、函数内联、常量表达式消除、死代码消除等
- -O2：这个级别会开启更多的优化，包括全局优化、自动对齐、软件流水线等。
- -O3：这个级别会开启所有可用的优化，除了那些可能会增加编译时间的优化。这包括 -O2 中的所有优化，以及一些额外的优化，如内联函数的优化、对数组访问的优化等。
- -Ofast：这个选项会开启所有优化，包括那些可能会增加编译时间的优化。这通常会生成最快的代码，但可能会牺牲代码的大小和可读性。
- -Os（最紧凑大小）：这个选项会优化代码的大小，而不是性能。编译器会尽可能生成小的代码，通常用于嵌入式系统或资源受限的环境。

编译优化标志

- -ffast-math：这个选项会关闭某些数学函数的严格准确性检查，以换取更高的性能，对浮点运算有帮助
- -funroll-loops：这个选项会展开循环，以减少循环控制的开销，对循环嵌套的程序有性能提升作用。
- -fomit-frame-pointer:这个选项用于决定是否在汇编代码中包含帧指针。帧指针是一个在函数调用时保存当前栈帧地址的寄存器。如果不使用帧指针，那么函数调用的开销会更小

GCC编译优化命令

- 编译命令示例：

```
gcc –O3 -fomit-frame-pointer myprogram.c -o myprogram
```

O3级别优化

不在汇编代码中包含帧指针

调整循环顺序

- 调整内外循环的顺序可能会影响代码的性能
  - 尤其是在涉及多维数组或矩阵操作时，这种影响通常与数据的访问模式和缓存利用率有关
  - 假设CPU缓存是按照行优先的方式工作的，那么行优先遍历可能会比列优先遍历更加高效，因为它减少了缓存失效的次数

```
// 行优先遍历
void row_major_traversal() {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            row_major_sum += matrix[i][j];
        }
    }
}
```

```
// 列优先遍历
void col_major_traversal() {
    for (int j = 0; j < N; ++j) {
        for (int i = 0; i < N; ++i) {
            col_major_sum += matrix[i][j];
        }
    }
}
```

循环展开（Loop Unrolling）

- 通过减少循环终止条件检查和循环索引更新的次数来提高代码性能
  - 这样可以减少循环的迭代次数、减少分支预测错误的可能性，从而提高代码的运行效率。

```
void loop_unrolling_example(int *arr, int size) {
    for (int i = 0; i < size; i += 4) {
        arr[i] += 1;
        arr[i+1] += 1;
        arr[i+2] += 1;
        arr[i+3] += 1;
    }
}
```

- Divide and conquer算法 – 便于并行
  - 直接思路：将每个矩阵分为4份
    - 8次小矩阵乘法和4次小矩阵加法计算
    - 时间复杂度$T(n) = 8T\left(\frac{n}{2}\right) + O(n^2) = O(n^3)$
      - 总计算量完全一致（参考分块计算）

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

A B C

Strassen算法优化

- Strassen算法
  - 仍然将矩阵A与B分别分为4块
    - 如下计算p1，p2，…，p7
    - 使用p1，p2，…，p7更新C中4块子矩阵的值，好处是?

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix}=\begin{pmatrix} p5+p4-p2+p6 & p1+p2 \\ p3+p4 & p1+p5-p3-p7 \end{pmatrix}$$

A　　B　　C

p1=a(f−h)
p2=(a+b)h
p3=(c+d)e
p4=d(g−e)
p5=(a+d)(e+h)
p6=(b−d)(g+h)
p7=(a−c)(e+f)

- **效率分析**：7次小矩阵乘法与18次小矩阵加法， 加法效率比乘法更高
- **时间复杂度**：$T(n)=7T\left(\frac{n}{2}\right)+O(n^2)=O\left(n^{log_2 7}\right)=O(n^{2.8074})$

Strassen算法优化

- **验证**：p5+p4-p2+p6=(a+d)(e+h)+d(g-e)-(a+b)h+(b-d)(g+h)

  =ae+ah+de+dh+dg-de-ah-bh+bg+bh-dg-dh

  =ae+bg

$$\underset{A}{\begin{pmatrix} a & b \\ c & d \end{pmatrix}}\underset{B}{\begin{pmatrix} e & f \\ g & h \end{pmatrix}}=\underset{C}{\begin{pmatrix} p5+p4-p2+p6 & p1+p2 \\ p3+p4 & p1+p5-p3-p7 \end{pmatrix}}$$

$p1=a(f-h)$

$p2=(a+b)h$

$p3=(c+d)e$

$p4=d(g-e)$

$p5=(a+d)(e+h)$

$p6=(b-d)(g+h)$

$p7=(a-c)(e+f)$

Intel MKL矩阵乘法

- 使用Intel MKL算法库来优化代码性能
  - 稠密矩阵乘法**cblas_dgemm**示范代码
    - https://www.intel.com/content/www/us/en/docs/onemkl/tutorial-c/2021-4/multiplying-matrices-using-dgemm.html

```
double *A, *B, *C;
int m, n, k, i, j;
double alpha, beta;

A = (double *)mkl_malloc( m*k*sizeof( double ), 64 );
B = (double *)mkl_malloc( k*n*sizeof( double ), 64 );
C = (double *)mkl_malloc( m*n*sizeof( double ), 64 );

cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
    m, n, k, alpha, A, k, B, n, beta, C, n);
```

Intel MKL矩阵乘法

- Intel oneAPI Math Kernel Library (MKL) – 使用
  - 稠密矩阵乘法**cblas_dgemm**示范代码
    - https://www.intel.com/content/www/us/en/docs/onemkl/tutorial-c/2021-4/multiplying-matrices-using-dgemm.html

```
double *A, *B, *C;
int m, n, k, i, j;
double alpha, beta;

A = (double *)mkl_malloc( m*k*sizeof( double ), 64 );
B = (double *)mkl_malloc( k*n*sizeof( double ), 64 );
C = (double *)mkl_malloc( m*n*sizeof( double ), 64 );

cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
    m, n, k, alpha, A, k, B, n, beta, C, n);
```

对比实验

- 实现多个版本串行矩阵乘法，并进行对比分析

版本	实现描述	运行时间（sec.）	相对加速比	绝对加速比	浮点性能（GFLOPS）	峰值性能百分比
1	Python					
2	C/C++					
3	调整循环顺序					
4	编译优化					
5	循环展开					
6	Intel MKL					

- “相对加速比”为相对于前一版本的加速比；
- “绝对加速比”为相对于版本1的加速比；
- “浮点性能”：统计程序里面跑了多少次浮点计算，然后除以运行时间
- “峰值性能百分比”为当前浮点性能相对于计算设备峰值性能的百分比

Questions?