# 并行计算 II 2025 春第二次作业
# 基于 OpenMP/MPI 实现迭代稀疏矩阵向量乘法

周瑞松　王飒

May 18, 2025

## 1　迭代稀疏矩阵向量乘法 (Iterative SpMV)

稀疏矩阵向量乘法 (SpMV, Sparse Matrix-Vector multiplication) 是科学计算和工程领域中的一个核心操作，广泛应用于求解线性方程组、特征值问题、图算法以及机器学习等。当矩阵非常大且大部分元素为零时，我们称之为稀疏矩阵。针对稀疏矩阵的 SpMV 操作 $y = Ax$ 需要特别的存储格式和算法以节省内存和计算时间。

本作业关注的是一种常见的迭代计算过程：

$$x_{k+1} = Ax_k + b$$

其中 $A$ 是一个给定的稀疏方阵，$x_k$ 是第 $k$ 次迭代的向量。迭代从一个初始向量 $x_0$ 开始，重复进行若干次。这种迭代模式是许多迭代求解器（如 Jacobi、Gauss-Seidel 方法的某些变体，或幂法求主特征向量）的基础。本次作业我们考虑 $b = x_0$，且 $x_0$ 第 $i$ 个元素值为 $(i-1)\%10$，$(i = 1, ..., n)$。

### 1.1　输入数据格式

为了简化输入处理，我们将采用自定义的文本格式存储稀疏矩阵 $A$，称为 ".csr" 格式。该格式直接存储了矩阵的 CSR (Compressed Sparse Row) 表示的关键数组：

- **第一行:** 包含三个整数，分别是 'num_rows' (矩阵行数), 'num_cols' (矩阵列数), 和 'num_nonzeros' (矩阵中非零元素的总数)，以空格分隔。
- **第二行:** 包含 'num_rows + 1' 个整数，代表 CSR 格式的 'row_ptr' 数组。这些值以空格分隔，并且是 **0-based** 索引。'row_ptr[i]' 指示第 $i$ 行 (0-indexed) 第一个非零元素在 'values' 和 'col_indices' 数组中的起始位置。'row_ptr[num_rows]' 等于 'num_nonzeros'。
- **第三行:** 包含 'num_nonzeros' 个整数，代表 CSR 格式的 'col_indices' 数组。这些值以空格分隔，并且是 **0-based** 索引，表示每个非零元素所在的列。
- **第四行:** 包含 'num_nonzeros' 个浮点数，代表 CSR 格式的 'values' 数组。这些值以空格分隔，表示每个非零元素的具体数值。

初始向量 $x_0$ 将在程序内部根据矩阵维度生成，例如，所有元素初始化为 $i\%10$ $(i = 0, ..., n-1)$。

### 1.2 算法基本流程 (CSR 格式下的 SpMV)

对于一次 SpMV 操作 $y = Ax$，当矩阵 $A$ 以 CSR 格式存储时，向量 $y$ 的第 $i$ 个元素计算如下：

$$y_i = \sum_{k=\text{row\_ptr}[i]}^{\text{row\_ptr}[i+1]-1} \text{values}[k] \times x[\text{col\_indices}[k]]$$

其中 $x$ 是输入向量。

**串行迭代流程**

**Input:** 稀疏矩阵 $A$ (从 ".csr" 文件读入并转换为 CSR 结构), 迭代次数 $N_{iter}$
**Output:** 最终向量 $x_{final}$ (写入文件), 执行时间

$x_{current} \leftarrow$ InitializeVector ▷ 初始化 $x_{current}$
$x_0 \leftarrow x_{current}$
**for** $iter \leftarrow 1$ to $N_{iter}$ **do** ▷ 进行 $N_{iter}$ 次迭代
    $x_{next} \leftarrow$ AllocateVector($A$.num_rows)
    **for** $i \leftarrow 0$ to $A$.num_rows $- 1$ **do** ▷ 对于矩阵的每一行
        $sum \leftarrow 0.0$
        **for** $k \leftarrow A$.row_ptr$[i]$ to $A$.row_ptr$[i+1] - 1$ **do** ▷ 遍历行 $i$ 的非零元
            $sum \leftarrow sum + A$.values$[k] \times x_{current}[A$.col_indices$[k]]$
        **end for**
        $x_{next}[i] \leftarrow sum$
    **end for**
    $x_{current} \leftarrow x_{next} + x_0$ ▷ 更新向量用于下一次迭代
**end for**
**return** $x_{current}$ ▷ 返回最终迭代结果

## 2 作业要求和提示

你需要开发三个版本的迭代 SpMV 程序：

1. **串行版本：** 名为 `serial.cpp` (或 `.cpp`)。
2. **并行版本：** `parallel.c` (或 `.cpp`)。

如果你希望使用其他语言（如 Fortran 等）请修改为你需要的后缀名。

- 我们提供一个 Python 脚本 (`check.py`) 用于比较两个输出向量的正确性。
- 提供的串行 C 代码 (`serial.c`) 实现了从 ".csr" 文件读取数据和迭代 SpMV 的逻辑，可作为你实现的基础或参考。你需要在此基础上或独立实现 OpenMP 或 MPI 版本。
- **集群环境：** 基于数学中心集群，最多使用 16 个进程；
- **编程语言与编译：** 推荐 C/C++ 实现。不限制编译选项和编译器版本，但请在报告中注明。

- **输入：**
  - 程序通过命令行参数接收输入的 ".csr" 文件名和迭代次数 $N_{iter}$。
  - 可选地，通过 '-o <output_filename>' 参数指定最终向量的输出文件名。
  - 我们将提供若干不同规模和稀疏度的 ".csr" 测试矩阵文件 ('input_1.csr', 'input_2.csr','input_3.csr', 'input_4.csr', 'input_5.csr', )。使用指令 "./serial inputx.csr -o outputx.txt" 进行测试。
  - $N_{iter}$ 设置为 3000。
- **输出：**
  - 如果指定了 '-o' 参数，则将最终迭代得到的向量 $x_{final}$ 输出到指定文件，每行一个元素，使用科学计数法并保证足够精度 (如 '%.15e')。
  - 程序应打印迭代计算部分的总执行时间。
  - 使用串行版本产生的结果作为参考标准。OpenMP 或 MPI 版本的输出向量应使用提供的 `check.py` 脚本与串行版本的结果进行比较，以验证正确性 (在浮点数容差范围内)。
- **计时范围：**主要对 $N_{iter}$ 次迭代的循环部分进行计时。数据读入、CSR 结构初始化、向量初始化以及最终结果写入文件的时间不应包含在主要性能计时内。对于 MPI 版本，迭代间的通信时间 (如 'MPI_Allgatherv') **应包含**在计时内。
- **报告要求：**
  - 对于 OpenMP 版本，说明并行区域、共享/私有变量、循环调度策略，以及如何合并结果。
  - 对于 MPI 版本，说明数据划分策略 (矩阵 $A$ 和向量 $x$ 如何划分给各进程)，以及迭代过程中向量 $x_k$ 的通信方式 (例如，是否使用 'MPI_Allgatherv' 来获取完整的 $x_k$)。
  - 对比串行、OpenMP (不同线程数) 和 MPI (不同进程数) 的执行时间，并计算加速比和并行效率。
- 鼓励测试不同规模的矩阵和不同的迭代次数，观察其对性能和加速比的影响。

# 3 提交要求和评分标准

- **提交文件：**
  - 你的 C/C++ 源代码文件：`serial.c` (或 `.cpp`), `parallel.c` (或 `.cpp`)，(如果你希望使用其他语言，请修改为你需要的后缀名)。
  - 一份报告（`report.pdf`），包含：
    - 你的姓名和学号。
    - **编译环境：**使用的编译器、版本及编译选项。
    - **串行版本描述：**简述实现，作为后续比较的基准。
    - **OpenMP 版本描述（若使用 MPI 请忽略）：**
      - 并行化策略（并行区域、共享/私有变量、循环如何并行化）。

- 使用的 OpenMP 指令和子句 (如 'schedule', 'reduction' 等)。
- 结果合并方式。

– **MPI 版本描述（若使用 OpenMP 请忽略）：**
- 数据划分策略（矩阵和向量如何分布）。
- 通信策略（例如，如何处理迭代中的 $x_k$ 向量，使用了哪些 MPI 函数）。
- 结果收集方式。

– **性能分析：**
- 一个或多个表格/图表，展示针对所选测试矩阵和不同迭代次数，串行、OpenMP (不同线程数，如 1, 2, 4, 8, 16) 或 MPI (不同进程数，如 1, 2, 4, 8, 16) 的执行时间。
- 计算并展示 OpenMP 或 MPI 版本的加速比 (Speedup) 和并行效率 (Efficiency)。
- 对性能结果的分析和讨论（例如，为什么加速比会饱和等）。

– **正确性验证：**简述如何使用提供的 `check.py` 脚本验证了并行版本的输出与串行版本的一致性（可以截图或摘录脚本输出的关键部分）。

–（可选）任何观察到的有趣现象、遇到的挑战或尝试的额外优化及其效果。

- **提交方式：**通过邮件提交你的文件 (打包成 zip 文件)。发送至 parco2025@163.com，邮件主题格式为："并行计算第二次作业提交-[你的姓名]-[你的学号]"，如 "并行计算第二次作业提交-周瑞松-2301110062"，zip 文件名与邮件主题相同。

- **截止日期： 2025 年 6 月 10 日 23:59**