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）

1-基于MPI的并行矩阵乘法

吴迪、刘学正

中山大学计算机学院

实验概要

- 内容
  - MPI程序编译、运行和调试
  - MPI点对点通信
  - MPI矩阵乘法程序性能分析

- 目标
  - 掌握 MPI 程序的编译/运行方法
  - 理解 MPI 点对点通信的基本原理
  - 了解MPI程序GDB调试流程

MPI程序基本结构

Hello world示例

0. 包含头文件

1. 初始化MPI环境，创建Communicator

2. 获取Communicator 的大小

3. 获取当前进程的 rank

4. 你的工作区域

5. 清理 MPI 环境

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

MPI点对点通信

邮局寄信的类比

- A→B： A 把信纸（数据）打包到一个信封（消息元数据）里，交给邮局（网络），邮局负责将信封送给B
- B：收到信封（消息元数据）后， B 确认里面是否有自己想要的信息，如果是，则取出信纸（数据）
- A：收到邮局（网络）送来的信息，得知B已经收到了信息

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

缓冲区

元素数量

数据类型

发送/接收方

消息标签

通信子

接收操作信息

Open MPI manual： https://docs.open-mpi.org/en/v5.0.x/man-openmpi/index.html

MPI点对点通信

乒乓程序示例：适用2个进程

```
int ping_pong_count = 0;
int partner_rank = (world_rank + 1) % 2;
while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
        // Increment the ping pong count before you send it
        ping_pong_count++;
        MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
        printf("%d sent and incremented ping_pong_count %d to %d\n",
               world_rank, ping_pong_count,
               partner_rank);
    } else {
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d received ping_pong_count %d from %d\n",
               world_rank, ping_pong_count, partner_rank);
    }
}
```

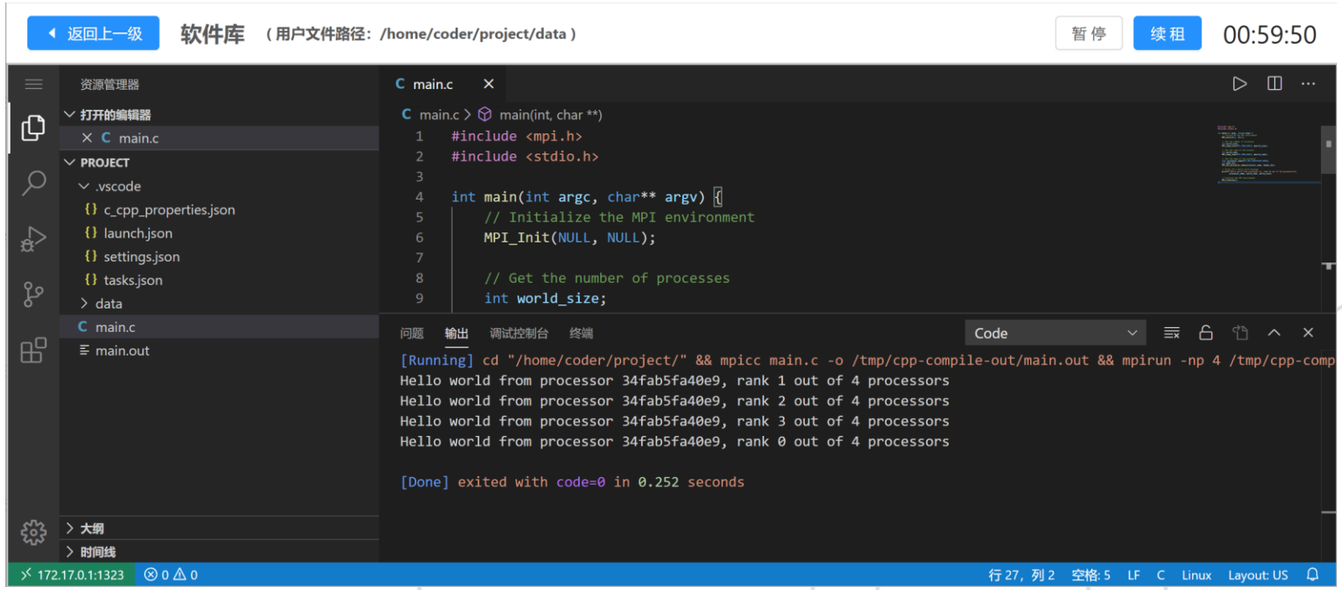
- 随着 ping_pong_count 的递增，两个进程会轮流成为发送者和接收者

MPI程序编译运行

- 命令行方式

```
mpicc -o hello_world hello_world.c
mpirun -n 4 ./hello_world
```

- 超算习堂VSCode
  - 软件库→并行编程Web-IDE



- 修改settings.json配置中的-np参数可运行不同进程数目
- 注意：参数修改后可能要重载VSCode，Ctrl+Shift+P→Developer: Reload Window

MPI程序调试

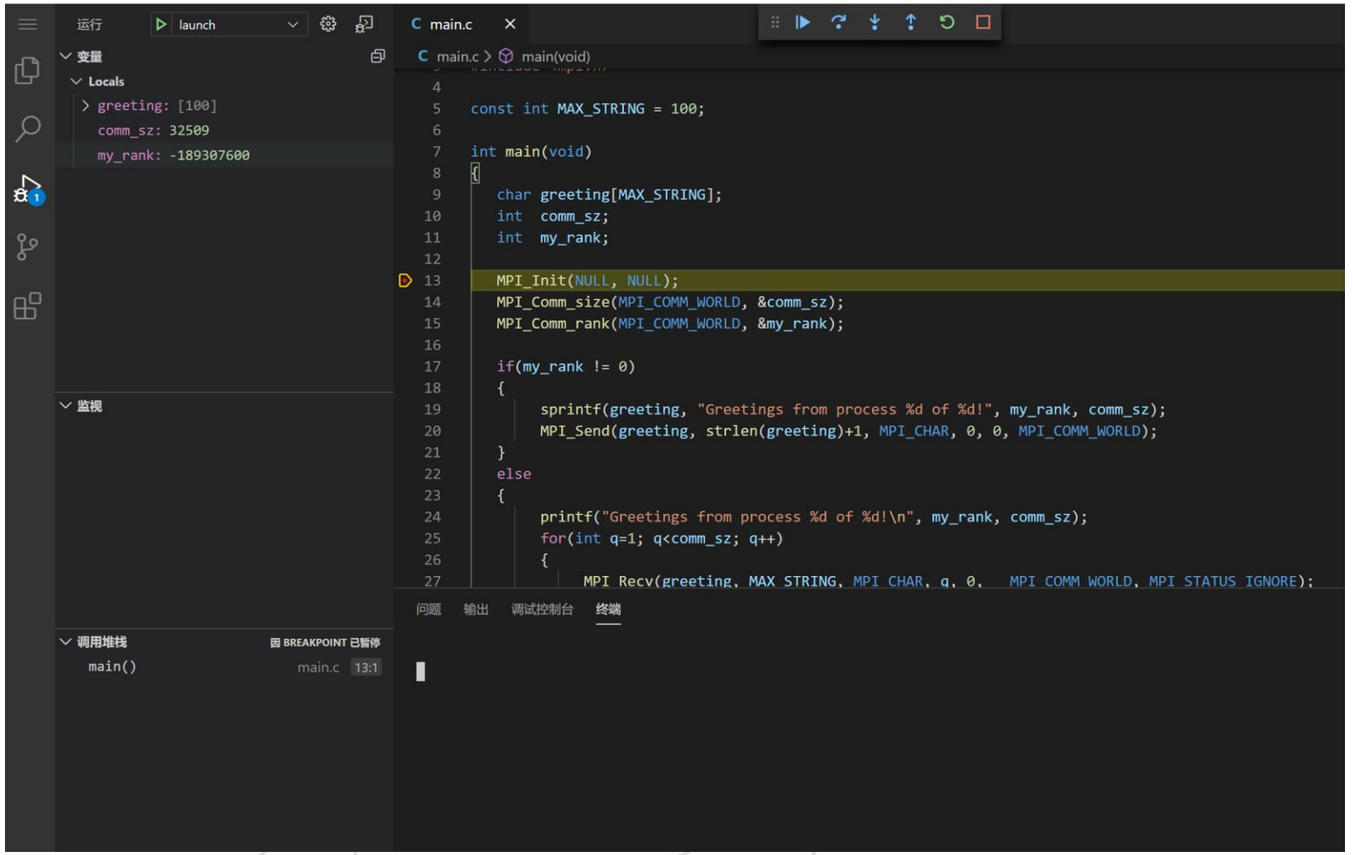
- GDB（GNU Debugger）是一个功能强大的**调试工具**，主要用于调试C、C++等编程语言编写的程序。它可以帮助开发者查找和修复代码中的错误。
- 调试模式：

特性	launch 模式	attach 模式
启动方式	启动新程序	附加到已运行的程序
程序生命周期	调试器控制程序从启动到结束	程序已运行，调试器只附加到现有进程
适用场景	从头调试程序	调试正在运行的程序
配置复杂度	较简单	需要指定进程 ID 或选择进程

MPI程序调试

单进程：VSCode中 GDB launch 调试模式

- 准备工作
  - 安装GDB：sudo apt-get install gdb
  - 安装 VSCode 的 C/C++ 扩展
- 配置调试环境
  - 创建launch.json文件
  - 配置构建任务（可选）：
    - 修改 "command": "mpicc“
- 开始调试
  - 设置断点
  - 进入调试视图，点击绿色启动按钮
  - 执行调试
    - 单步执行：使用 F10（跳过）或 F11（进入函数）
    - 使用 F5 继续到下一个断点



Debug a Parallel MPI program in Visual Studio Code：https://www.bilibili.com/video/BV1SQ4y1Q7R1/

MPI程序调试

- 多进程：VSCode中 GDB attach 调试模式
  - 准备工作、配置调试环境与单进程类似
  - 修改调试配置文件（ launch.json ）
    - "request":"attach"
    - "processId": "${command:pickProcess}"
    - 几个进程就复制几个configurations
  - 在程序开头加入死循环

```
int main(int argc, char** argv) {

  int j=1;
    while(j){
    sleep(5);
  }
```

```
vscode > {} launch.json > Launch Targets > {} launch2
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "launch",
            "type": "cppdbg",
            // "request": "launch",
            "request":"attach",
            "processId": "${command:pickProcess}",
            "program": "${fileDirname}/${fileBasenameNoExtension}.out",
            "internalConsoleOptions": "neverOpen",
            "MIMode": "gdb",
            "miDebuggerPath": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "build",
        },
```

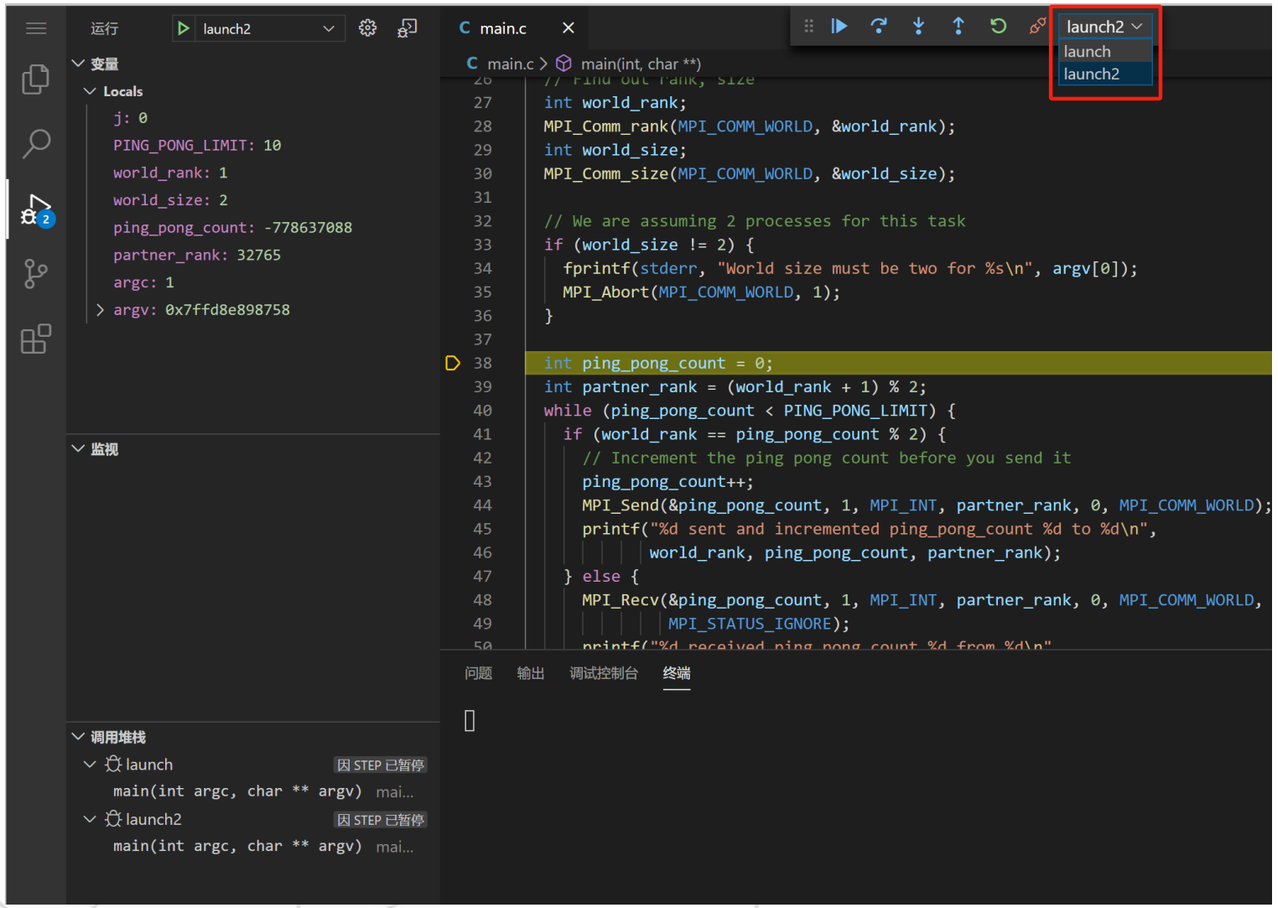
  - 编译程序（假设构建任务已在tasks.json定义）
    - 终端→运行生成任务→ build（Ctrl+Shift+B）

Debug a Parallel MPI program in Visual Studio Code：https://www.bilibili.com/video/BV1SQ4y1Q7R1/

MPI程序调试

多进程： VSCode中 GDB attach 调试模式

- 启动目标程序
  - mpirun -np 2 ./main.out
- 开始调试
  - 设置断点
  - 进入调试视图，启动调试选择要附加的进程
  - 暂停循环块，并设置j=0，跳出循环代码块
  - 切换launch配置，分别调试不同进程



Debug a Parallel MPI program in Visual Studio Code：https://www.bilibili.com/video/BV1SQ4y1Q7R1/

实验内容

- 要求
  - 使用MPI点对点通信实现并行矩阵乘法
  - 设置进程数量（1~16）及矩阵规模（128~2048）
  - 根据运行时间，分析程序并行性能

进程数	矩阵规模				
	128	256	512	1024	2048
1					
2					
4					
8					
16					

实验内容

- 讨论题
  - 在内存受限情况下，如何进行大规模矩阵乘法计算?
  - 如何提高大规模稀疏矩阵乘法性能?

Questions?