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）

2-基于MPI的并行矩阵乘法（进阶）

吴迪、刘学正

中山大学计算机学院

MPI集合通信

广播操作

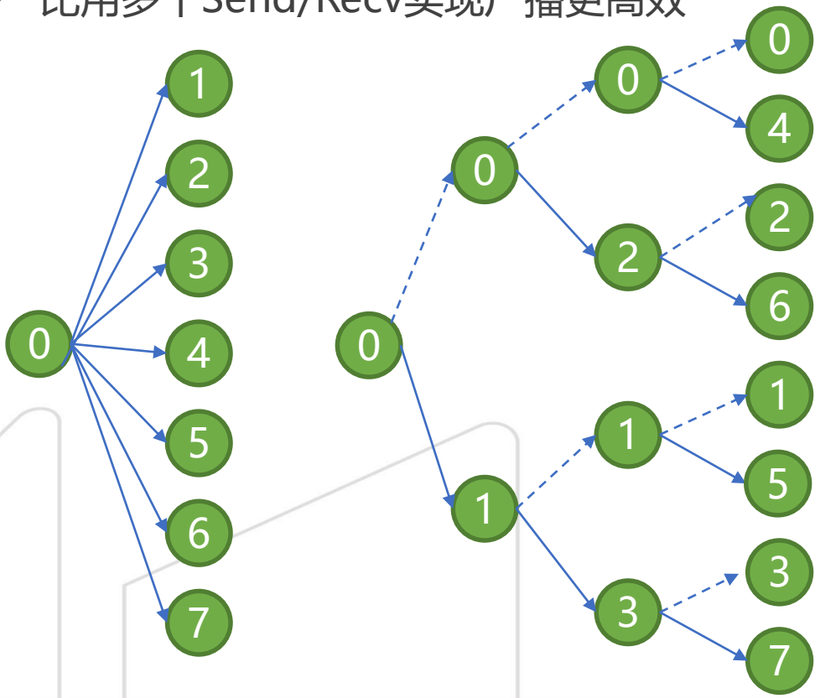
– MPI_Bcast：一个进程将**同样一份数据**传递给一个communicator里的所有其他进程

```
int MPI_Bcast(
    void *buffer,        // 数据缓冲区指针
    int count,           // 数据元素个数
    MPI_Datatype type,   // MPI数据类型
    int root,            // 根进程的rank
    MPI_Comm comm        // 通信器
);
```

所有进程调用相同的 MPI_Bcast：根进程的 buffer 向外广播，其他进程的 buffer 接收数据

效率

- MPI实现会优化广播算法（如使用树形广播）
- 比用多个Send/Recv实现广播更高效

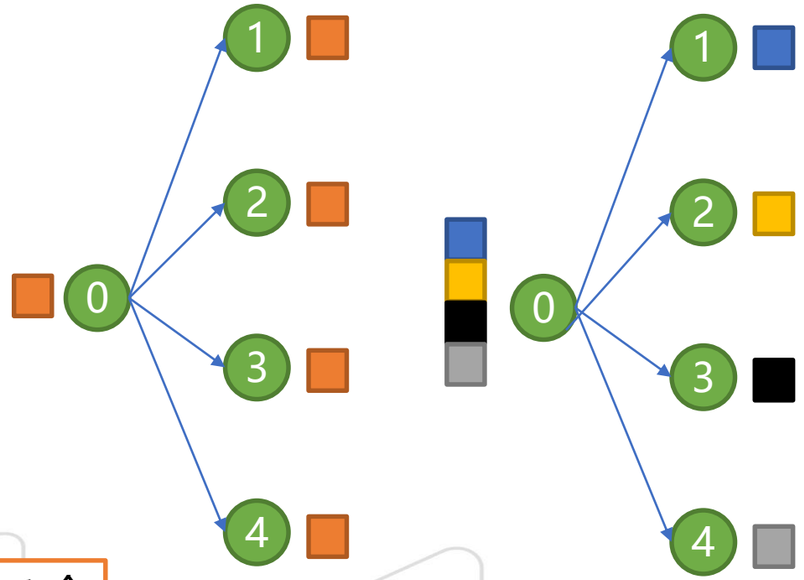


MPI集合通信

散射操作

– MPI_Scatter：一个进程将**一份数据不同部分**传递给一个 communicator 里的所有其他进程

```
int MPI_Scatter(
    const void *sendbuf,  // 根进程的发送缓冲区（仅根进程有效）
    int sendcount,        // 发送给每个进程的数据量
    MPI_Datatype sendtype,// 发送数据的 MPI 数据类型
    void *recvbuf,        // 接收缓冲区（所有进程）
    int recvcount,        // 接收的数据量（通常等于 sendcount）
    MPI_Datatype recvtype,// 接收数据的 MPI 数据类型
    int root,             // 根进程的 rank
    MPI_Comm comm         // 通信器（如 MPI_COMM_WORLD）
);
```



- 根进程（root）：提供一个数组 sendbuf，其中包含 **comm_size × sendcount** 个数据元素。该数组将被均匀拆分，每个子进程接收 **sendcount** 个元素。
- 其他进程（非 root）：仅需提供 recvbuf 接收缓冲区，sendbuf 参数可设为 NULL。接收来自根进程的 recvcount 个数据

MPI集合通信

收集操作

- MPI_Gather：一个 communicator 里的所有其他进程将**一份数据不同部分**传递给一个进程

```
int MPI_Gather(
    const void *sendbuf,   // 发送缓冲区（每个进程提供的数据）
    int sendcount,         // 每个进程发送的数据量（以元素为单位）
    MPI_Datatype sendtype,// 发送数据的 MPI 数据类型
    void *recvbuf,         // 接收缓冲区（仅根进程有效）
    int recvcount,         // 每个进程的数据量（通常等于 sendcount）
    MPI_Datatype recvtype,// 接收数据的 MPI 数据类型
    int root,              // 根进程的 rank
    MPI_Comm comm          // 通信器（如 MPI_COMM_WORLD）
);
```

如果不同进程发送的数据量不同，怎么办？

```
int MPI_Gatherv(
    const void *sendbuf,      // 发送缓冲区（每个进程）
    int sendcount,            // 本进程发送的数据量
    MPI_Datatype sendtype,    // 发送数据类型
    void *recvbuf,            // 接收缓冲区（仅根进程有效）
    const int *recvcounts,    // 各进程的数据量数组（长度为comm_size）
    const int *displs,        // 各进程数据在recvbuf中的偏移量数组
    MPI_Datatype recvtype,    // 接收数据类型
    int root,                 // 根进程rank
    MPI_Comm comm             // 通信器
);
```

- 根进程（root）：提供 recvbuf，其大小必须至少为 **comm_size × recvcount** 个元素，每个子进程发送 **recvcount** 个元素
- 其他进程（非 root）：仅需提供 sendbuf，其中包含 **sendcount** 个数据元素，recvbuf 参数可设为 NULL

MPI派生数据类型

- MPI派生数据类型（Derived Datatypes）允许用户自定义复杂的数据结构，以便在通信中高效传输**非连续或异构数据。**
- MPI_Type_create_struct 是**最灵活**的数据类型创建函数，允许将不同基本类型组合成一个新的派生数据类型

```
int MPI_Type_create_struct(
    int count,                                  // 块的数量
    const int array_of_blocklengths[],          // 每个块的元素个数
    const MPI_Aint array_of_displacements[],    // 每个块的字节偏移量
    const MPI_Datatype array_of_types[],        // 每个块的数据类型
    MPI_Datatype *newtype                       // 输出的新数据类型
);
```

使用方式

- 创建数据类型：newtype=MPI_Type_create_struct(...)
- 提交数据类型：MPI_Type_commit(&newtype)
- 使用数据类型：在通信操作中使用
- 释放数据类型：MPI_Type_free(&newtype)

```
typedef struct {
    int id;         // 4字节
    double value;   // 8字节
    char tag;       // 1字节
} MyStruct;
```

```
+-------------------------------------------------------------------+
| MPI_Type_create_struct 参数布局                                   |
|                                                                   |
| count = 3                                                         |
|                                                                   |
| array_of_blocklengths: [1,      1,       1]                       |
|                         (id)    (value)  (tag)                    |
|                                                                   |
| array_of_displacements: [0,      4,       12]  (字节偏移量)       |
|                           |_____|________|                        |
|                            id    value    tag                     |
|                                                                   |
| array_of_types: [MPI_INT, MPI_DOUBLE, MPI_CHAR]                   |
|                                                                   |
+-------------------------------------------------------------------+
```

参数图解

实验内容

- 实现方式
  - 使用MPI集合通信实现并行矩阵乘法
  - 使用MPI_Type_create_struct聚合进程内变量后通信
  - 选做：尝试不同数据/任务划分方式
- 结果分析
  - 设置线程数量（1-16）及矩阵规模（128-2048）
  - 根据运行时间，分析程序并行性能及扩展性

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

Questions?