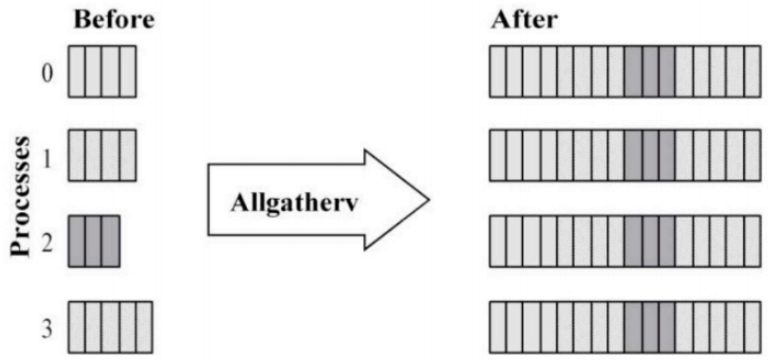
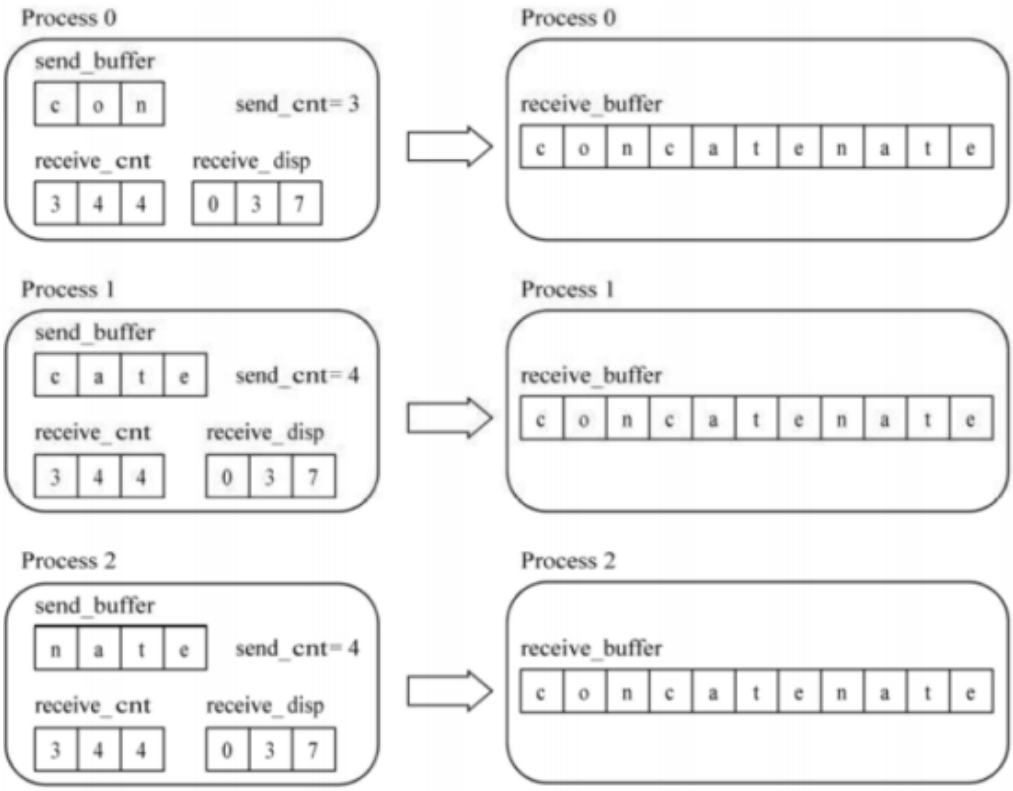
9 MPI编程——矩阵向量乘法&MPI+OpemMP

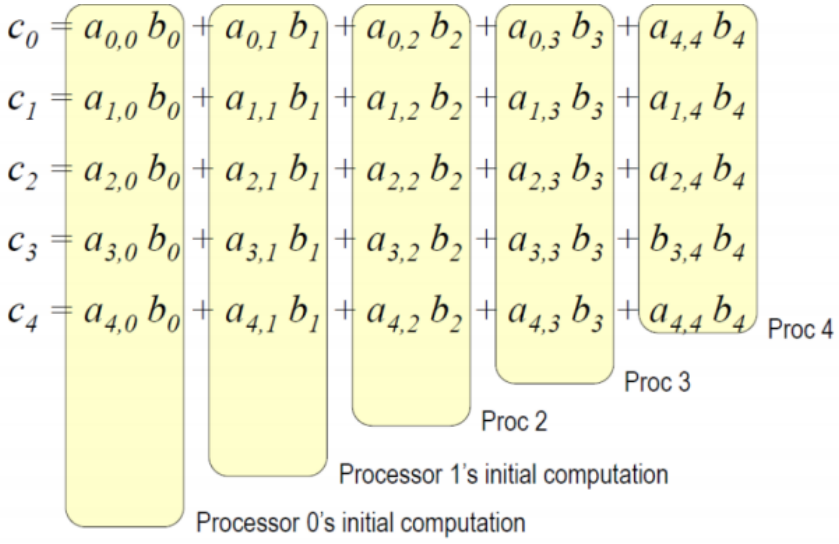
- 大纲
  - 例子：矩阵向量乘法
    - 串行算法
    - 三个并行程序的设计、分析和实施；
      - 逐行块条纹
      - 逐列块条纹
      - 棋盘格
  - MPI+OpenMP
- 矩阵向量乘法
  - 逐行分块
    - 通过域分解进行分区
    - 与——相关的原始任务
      - 矩阵行
      - 整个向量
    - 将A各行拆分，b不动，各个进程完成两个向量的内积得到c_i
    - int MPI_Allgatherv (void *send_buffer, int send_cnt, MPI_Datatype send_type, void *receive_buffer, int *receive_cnt, int *receive_disp, MPI_Datatype receive_type, MPI_Comm communicator)——将不同长度的向量聚集到一起





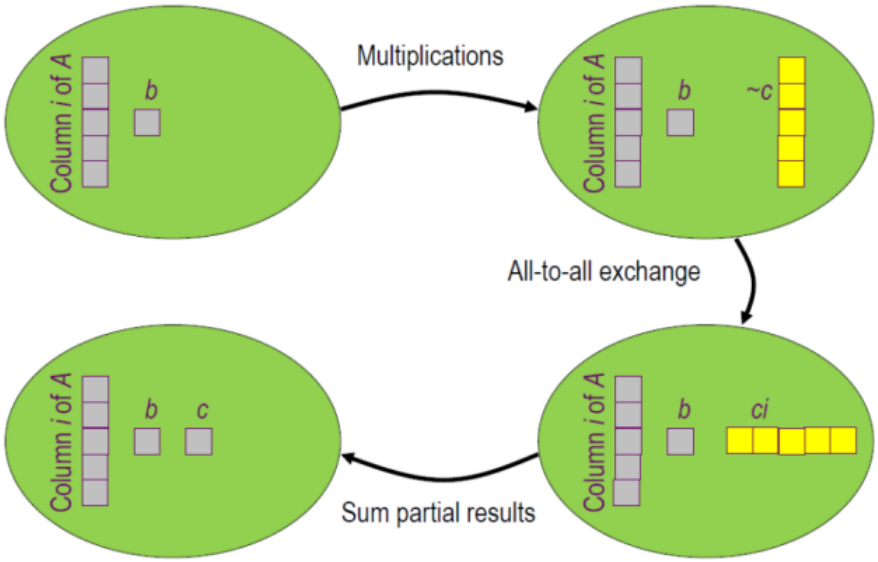
- 集聚与映射
  - 静态任务数
  - 定期沟通模式(all-gather)
  - 每个任务的计算时间是恒定的
  - 战略：
    - 聚合行组
    - 每个 MPI 进程创建一个任务

- 逐列分解矩阵

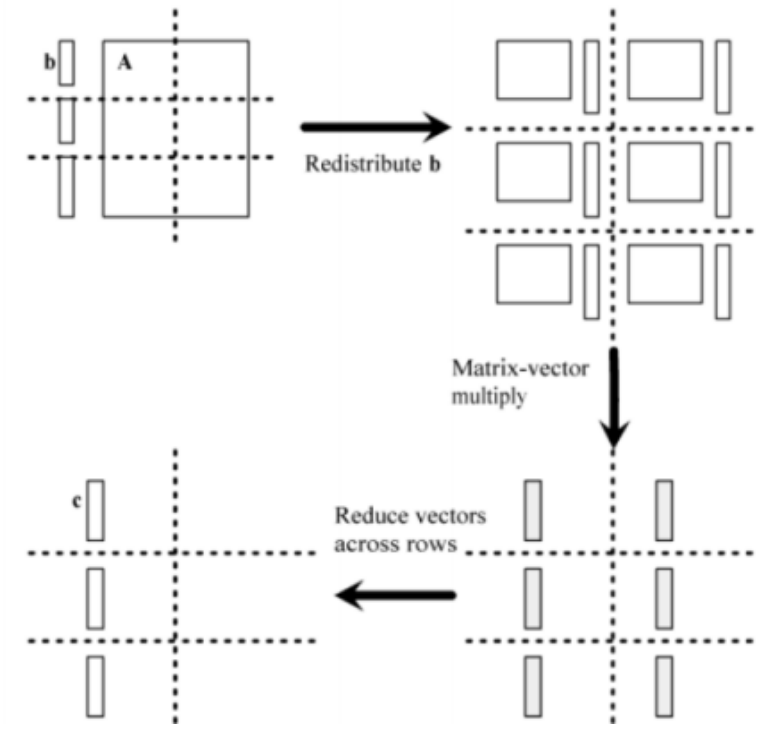


- 调用 MPI_Alltoall(...)传送计算结果。其中每个进程完成计算后得到上图中 $c_{0j}, c_{1j}, c_{2j}, c_{3j}, c_{4j}$。最后分组加和。
- 通过域分解进行分区
- 相关的任务

    - 矩阵列
    - 向量元素
  - 流程图

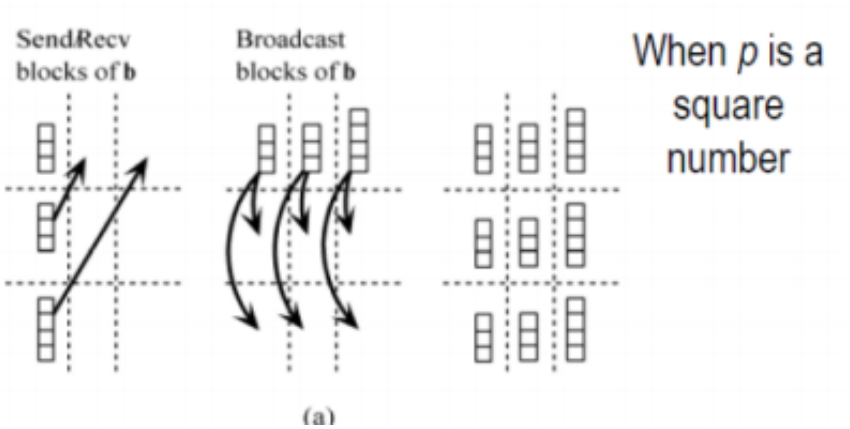


  - 聚集和映射
    - 静态任务数
    - 定期沟通模式（all to all）
    - 每个任务的计算时间是恒定的
    - 战略：
    - 聚集列组
    - 每个 MPI 进程创建一个任务
- 棋盘格分解

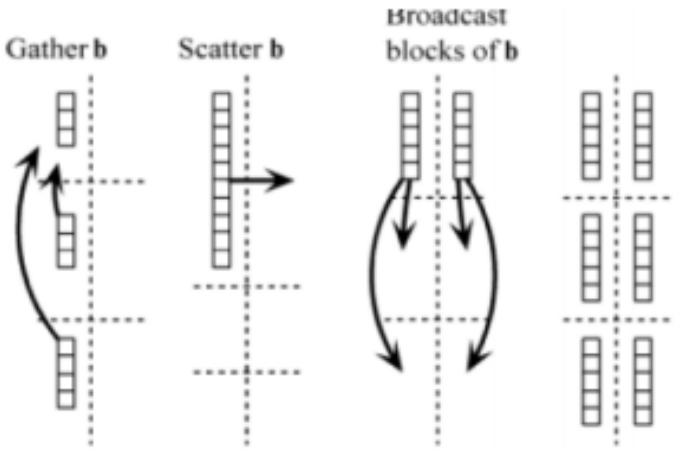


  - 将原始任务与矩阵 A 的每个元素相关联

- 每个原始任务执行一个乘法
- 将原始任务聚合成矩形块
- 流程形成二维网格
- 向量 b 按块在网格第一列中的进程之间分布
- 划分棋盘格流程重新分配向量 b
  - 步骤 1：将 b 从第一行的进程移动到第一列的进程
    - 如果 p 是一个数的平方：第一列/第一行进程发送/接收部分 b

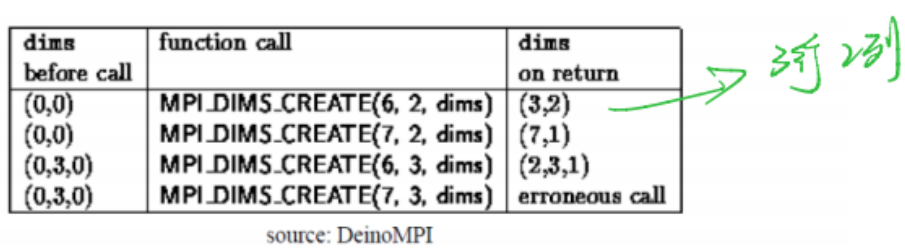


    - 如果 p 不是是一个数的平方：
      - 在进程 0, 0 上收集 b
      - 处理 0, 0 个散发到第一行 procs



  - 第 2 步：第一行进程在列内广播 b
- 创建通信子
  - 想要虚拟二维网格中的进程
  - 创建一个自定义通信子来执行此操作
  - 集体通信涉及通信子中的所有进程
  - 我们需要在进程子集之间进行广播、缩减
  - 我们将为同一行或同一列中的进程创建通信子
- 通信子中有什么？
  - 进程组

    - 上下文
    - 属性
        - 拓扑（让我们以另一种方式处理进程），rank--线性结构，空间坐标--拓扑非线性
- 创建流程的二维虚拟网格
    - MPI_Dims_create(int nodes, int dims, int* size)
        - 输入参数
            - 所需网格中的进程总数
            - 网格尺寸数（维度）
        - 返回每个维度中的进程数



dims before call	function call	dims on return
(0,0)	MPI_DIMS_CREATE(6, 2, dims)	(3,2)
(0,0)	MPI_DIMS_CREATE(7, 2, dims)	(7,1)
(0,3,0)	MPI_DIMS_CREATE(6, 3, dims)	(2,3,1)
(0,3,0)	MPI_DIMS_CREATE(7, 3, dims)	erroneous call

source: DeinoMPI

    - MPI_Cart_create(MPI_Comm old_comm, int dims, int* size, int* periodic, int reorder, MPI_Comm* cart_comm)
        - 创建具有笛卡尔拓扑结构的通信子
    - 使用

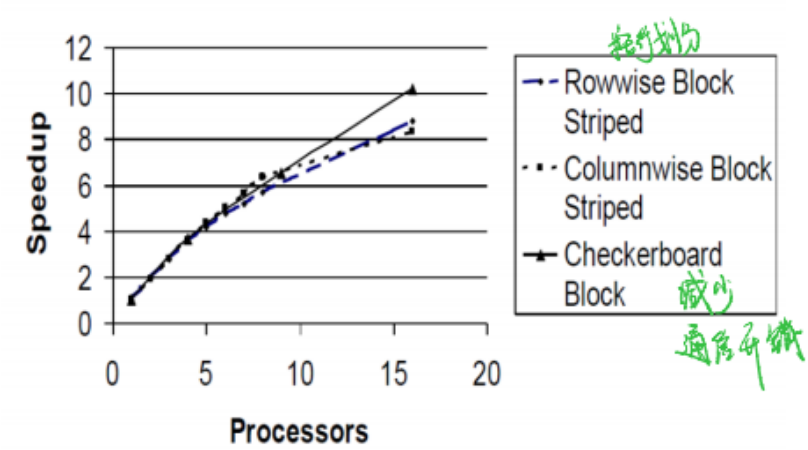
Using MPI_Dims_create and MPI_Cart_create

```
MPI_Comm cart_comm;
int p;
int periodic[2];
int size[2];
...
size[0] = size[1] = 0;
MPI_Dims_create (p, 2, size);
periodic[0] = periodic[1] = 0;
MPI_Cart_create (MPI_COMM_WORLD, 2, size,
1, &cart_comm);
```

    - 相关函数
        - MPI_Cart_rank(MPI_Comm comm, int* coords, int* rank)：给定笛卡尔通信子中的进程坐标，返回进程等级
        - MPI_Cart_cords(MPI_Comm comm, int rank, int dims, int *coords)：给定笛卡尔通信子中的进程等级，返回进程的坐标
    - 集合通信

Operation	MPI Name
One-to-all broadcast	MPI_Bcast
All-to-one reduction	MPI_Reduce
All-to-all broadcast	MPI_Allgather
All-to-all reduction	MPI_Reduce_scatter
All-reduce	MPI_Allreduce
Gather	MPI_Gather
Scatter	MPI_Scatter
All-to-all personalized	MPI_Alltoall

- 三种划分方式比较



- MPI+OpenMP
  - **优势**：
    - 降低通信开销
      - 使用 mk 进程传递消息 vs 使用 m 个进程进行消息传递，每个进程有 k 个线程
    - 程序的更多部分可以并行化
    - 可能允许更多的通信与计算重叠
      - 例如，如果 3 个 MPI 进程正在等待消息，并且 1 个 MPI 进程处于活动状态，则值得 fork 一些线程来加速第 4 个进程
  - 矩阵向量乘法

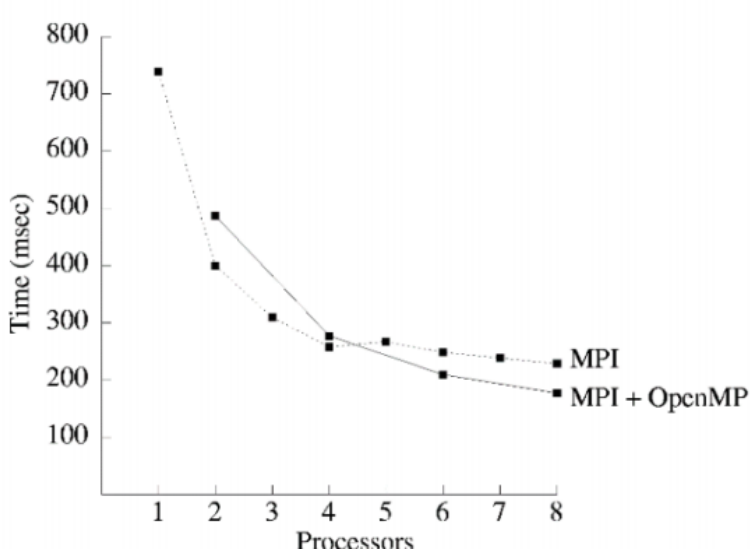
```
void matrix_vector_product (int id, int p,
   int n, double **a, double *b, double *c)
{
   int    i, j;
   double tmp;       /* Accumulates sum */
   for (i=0; i<BLOCK_SIZE(id,p,n); i++) {
      tmp = 0.0;
      for (j = 0; j < n; j++)
         tmp += a[i][j] * b[j];
      piece[i] = tmp;
   }
   new_replicate_block_vector (id, p,
      piece, n, (void *) c, MPI_DOUBLE);
}
```

- 添加OpenMP指令
  - 希望通过使并行成为可能的最外层循环来最小化 fork/join 开销
  - 如果每个线程都有 tmp 和 j 的私有副本，则可以并行执行外循环

```
#pragma omp parallel for private(j,tmp)

  for (i=0; i<BLOCK_SIZE(id,p,n); i++) {
```

  - 线程的用户控制
    - 希望让用户有机会指定每个进程的活动线程数
    - 向函数 main 添加对 omp_set_num_threads 的调用
    - 参数来自命令行：omp_set_num_threads (atoi(argv[3]));
- 基准测试
  - 目标系统：具有四个双处理器节点的商品集群
  - MPI 程序在 1、2、...、8 个 CPU 上执行
    - 在 1、2、3、4 个 CPU 上，每个进程在不同的节点上，最大化每个 CPU 的内存带宽
  - MPI+OpenMP程序在1、2、3、4进程上执行
    - 每个进程有两个线程
  - C+MPI+OpenMP程序在2、4、6、8线程上执行



- 对结果的分析
  - MPI+OpenMP 程序在 2、4 个 CPU 上速度较慢，因为 MPI+OpenMP 线程共享内存带宽，有竞争，而 C+MPI 进程不
  - MPI+OpenMP 程序在 6、8 个 CPU 上更快，因为它们具有更低的通信成本
- 案例研究：雅可比方法
  - 从使用 Jacobi 方法解决稳态热分布问题的 MPI 程序开始
  - 基于包含有限差分网格（FDM）的二维矩阵的行块条带分解程序
- 高斯消元
  - 用于解决 A 是稠密矩阵时的 Ax = b
  - 将 Ax = b 简化为上三角系统 Tx = c
  - 然后反向替换可以解决 Tx = c for x
  - 稀疏系统
    - 高斯消除不适用于稀疏系统
    - 系数矩阵逐渐填充非零元素
    - 增加存储要求
    - 增加总操作数
  - 迭代方法
    - 迭代法：生成一系列近似解值的算法
    - 比直接方法需要更少的存储空间
    - 由于它们避免对零元素进行计算，因此可以节省大量计算
  - 即使使用 Jacobi 方法，收敛速度也往往太慢而无法实用
  - 我们将继续使用收敛速度更快的迭代方法
    - MPI程序的概要文件执行
    - 专注于向大多数计算密集型函数添加 OpenMP 指令
- 总结
  - 混合 C+MPI+OpenMP 程序比 C+MPI 程序更快

- 混合程序的计算/通信比优越
- 对于混合程序，每个节点通信的网格点数是每个节点的两倍
- 较低的通信开销可在 8 个 CPU 上提高 19% 的加速

以上内容整理于 幕布文档