©中国科学院软件研究所版权所有.

HPL 中行交换算法

(并行软件与计算科学实验室, 软件研究所, 中国科学院, 北京 100190, 中国)

# 标准程序中的行交换算法

* 1. 行交换指标计算

根据panel分解得到的dpiv指标，可以计算出每个进程行需要交换的数据行下标。

由于行交换过程不可能用全部串行的低效方法去实现，所以需要重新计算指标。

Dpiv中存放的是global index.

首先，要把这些global index按0~nb重排。这样避免出现进程间重复发送消息。

交换算法指标计算方法如下：

1. 现在的nb个指标，写成nb个“交换对”，即(0, dpiv[0]), (1, dpiv[1]),。。。。。。在初始阶段，先加入第一个对的两次操作(0, dpiv[0]), (dpiv[0], 0), (1, dpiv[1]), (dpiv[1], 1)。
2. 如果下一个交换对的source和dst都是前面从未出现过的，那么就直接加这样2对指标。如果下一个source在前面的某个dst里出现了，那么就用dst替换前面的dst; 如果下一个dst在前面的某个dst里出现了，那么就用source替换前面的dst。也就是说，交换过程要保证每个数被写到自己最终要写入的位置。
3. Lst是写入位置为src的交换对。把它与src位置的交换对相交换。这样保证写入的前nb个下标是按顺序排列的。

例如，考虑下面的序列：0-3, 1-2, 2-3，3-3. 最后，应该是3，2，0,1的顺序。所以交换序列就写成（3,0），（1,3），（0,2），（2,1）。按算法顺序，先是（3,0），（0,3）。然后(3,0)，（0，3），（2,1），（1,2）。这时lst是4，i\*2是2，所以交换成(3,0)，（2,1），（0，3），（1,2）。下一步，2出现在dst里，变成(3,0)， （2,1），（0，3），（1,3）; 3也出现在dst里，就变成(3,0)，（2,1）， （0，2），（1,3）。这时lst是4，而i\*2=4,所以不需要交换，保持(3,0)，（2,1）， （0，2），（1,3）。下一步，src和dst一样，而且在前面出现了，所以不需要增加，只需要交换。lst是6，而i\*2=6，所以交换的操作结果保持前一步的状态。这个序列就是最终IPID序列，其中每一对的第一个值是要读的数的位置，第二个值是要写入的位置。

* 1. Binary-exchange算法

第一步，执行HPL\_lindx0().流程如下：

对每一个交换对IPID[i]，IPID[i][0]是需要读取的位置src，IPID[i][1]是需要写入的位置dst。计算对应的下标IPID[i][0] 所在的进程行。如果在本地，找到该行的本地下标，存入LINDXA[i]。如果本进程是非当前行进程，那么说明该行肯定要换入全局U的行，所以直接记录LINDXAU[i] = dst - ia;

如果本进程是当前行进程，那么要考查IPID[i][1], 也就是写入位置。如果要写入的行在本地进程，分以下两种情况考虑。一是要写入U的，一是要从U换出到A（非U部分）的。前者是正数，后者取反，也就是说，得到的下标是负数。

如果要写入的行在其他进程，找到需要读取该行数据的交换对，将LINDXAU设为此交换对的dst下标(ipid[i][1])，即在U中的下标。

此外，将每个进程贡献到U的行数记录在LLEN中。这样每个进程都知道各个进程要换出多少行。

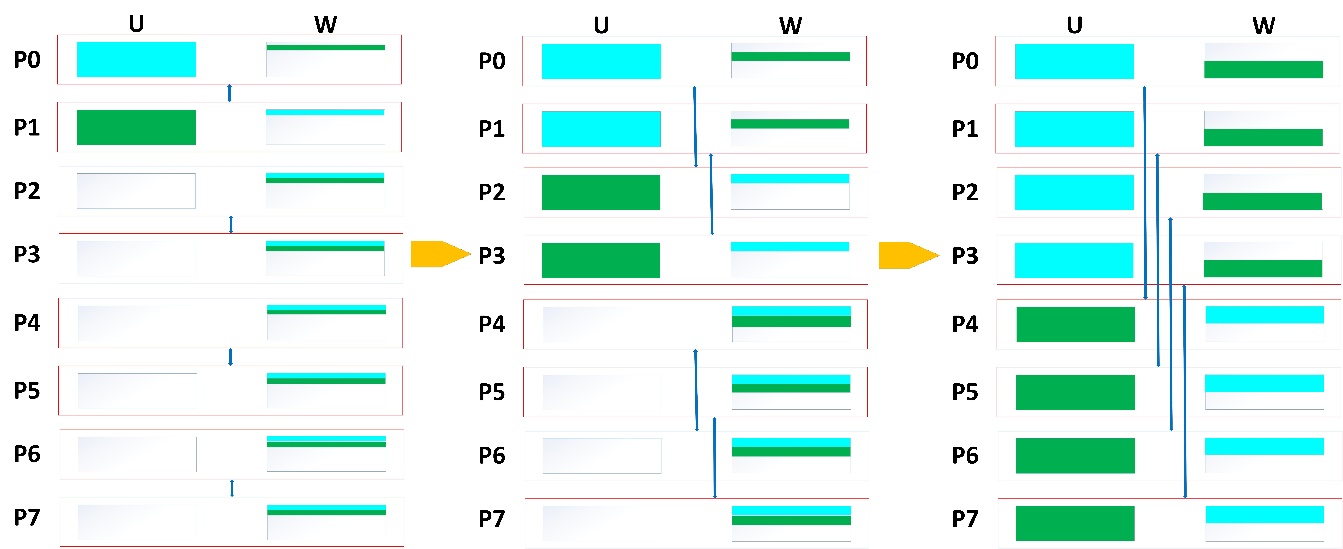
第二步，开始交换。

首先，当前行运行HPL\_dlaswp01T, 把需要换出的数据放到U中, 并把本进程内需要交换的数据进行交换。非当前行执行HPL\_dlaswp02N, 把需要换出的数据放到W中。

然后，If nprow is not a power of 2, proc[i-ip2] receives local data from proc[i] for all i in [ip2..nprow); icurrow is the source, these last process indexes are relative to icurrow.并更新llen。

然后按以下步骤做log(nprow)次交换。每次交换分前段进程和后段进程，以K为界点。K从2开始，每次交换之后增倍。在每一次交换中，前K个进程交换U。后面的nprow-K个进程只交换W。交换过程如下图所示。在前K个进程中，前K/2个进程向后面的K/2个进程发送完整的U，即nb行。与此同时，前K/2个进程接收后K/2个进程发送的W，也就是后K/2个进程需要换出的行。后面的nprow-K个进程，只交换W。交换之后，前K个进程需要进行本地拷贝。前K/2个进程要将接收到的W里的数据写入U中相应的行。后K/2个进程要将本地需要换出的数据写入U中相应的行。这样一来，前K个进程都持有相同的（已增加的）U了。然后，要更新llen，因为下一步要传递的数据包需要包括现在已接收的数据。

在下图中，可以注意到，在两个进程交换W之后，它们的W并不是一致的。但是这不妨碍它们被写入U时能写到正确的位置。因为在W的第一列，保存了它在U中应被写入的位置。



最后，如果nprow不是2的幂，还要再做一次消息传递。

* 1. Spread-roll 算法

第一步，执行HPL\_lindx1().流程如下：

首先，执行HPL\_lindx10()。根据ipid，计算出每个进程贡献给U的行数。对非当前行进程，只要dst所在进程不是本进程，那么就是当前行进程，需要计数；对当前行进程，只要dst的下标在jb范围内，那么就是要贡献给U的，需要计数。然后执行HPL\_logsort（）。该函数将所有进程按需要换出的数据量排序。也就是说，需要换出的行越多，越要排在离当前行近的位置。这样排序的目的是减少总的通信量。

然后与HPL\_lindx0()类似，计算LINDXA和LINDXAU。不同的是，此处还要计算permU。因为在binary-exchange算法中，HPL\_lindx0算出的LINDXAU即是每一行在U中的最终位置。而HPL\_lindx1（）计算出的LINDXAU，把每一进程所贡献的行，都集中在一起。因此，当通信过程结束后，每个进程都要在本地执行一次置换操作，将U中的各行放到正确的位置。

HPL\_lindx1计算过程如下。

对非当前行进程来说，考察每一个ipid[i]。如果dst在本进程，那么放到LINDXA中（即需要从U换入到本地的行）。如果src在当前行进程，那么有两种情况。第一种，是当前行进程内部交换的，那么，需要设置permU为dst。第二种，是当前行进程换出到其他行进程的（dstrow!=icurrow），那么，要找到这个dst是从哪一行换过来的（src），permU设为src，也就是说，这一行要写入U的src位置.

然后，执行HPL\_perm()。此函数将permU重排，使之能够完成原地置换。

第二步，开始交换。

首先，是spread过程，当前行进程将需要换出的行分发给其他所有进程，相当于MPI\_Scatter的操作，如下图所示。图中的进程号是HPL\_logsort（）排序之后的进程号，纵轴是时间轴，也就是说，一共需要log(P)次传输。将数据传到最终的目标进程最多需要log(P)跳。绿线所画的，是跳数的分界线。如前所述，HPL\_logsort()使得需要数据交换最少的进程排在了跳数最多的位置。

假设每一列有n个进程。第一次传输时，P0要把所有需要与(Pn/2到P n-1这n/2个进程)交换的行（即需要与最后n/2个进程交换的行）发送给P n/2。第二次传输时，P0把需要与(P n/4-P n/2-1)交换的行发送给P n/4，P n/2把需要与(P 3n/4-P n-1)交换的行发送给P 3n/4。依此类推。

然后, 非当前列进程执行HPL\_dlaswp06T, 也就是把U中的数据与本地需要换出的数据交换.

这时,每个进程的U中已经有了本进程要贡献到U中的一部分. 所以下面是roll过程. 此过程类似panel广播中long算法. 总共需要进行n-1次传输.每次传输,所有的进程都要参与, 每个进程只与自己的邻居交换. 第一步,先把自己所拥有的一部分与左(右)邻居交换, 然后,把自己刚刚得到的数据与右(左)邻居交换. 这样左右交替,直到所有的数据都交换完毕,每个进程都拥有了全部的U.

下面,所有进程还需要执行HPL\_dlaswp10N，也就是把所有的数据根据之前得到的permU序列进行重排，放它们放到应有的位置。

