# 戴新颜 28026598

# 实现

1. 文件IO

读文件：

每个节点根据元素个数平均分配(number\_of\_elements/number\_of\_node); 然后直接通过MPI\_IO读取所分配的elements。

写文件：各节点通过MPI\_IO把排序后elements直接写入相应的位置。

1. basic version

基本的even-odd排序，

在每一个排序phase中，每一节点通过与左/右节点交换末端元素，完成一个phase。

完成一个phase便判断本节点所有元素是否有交换，并且通知节点0.

节点0手机所有节点交换信息并且合并（只要有一个点仍然在交换即可以判断排序为结束），节点0再通知所有节点。

其他节点收到节点0的信息后判断是否继续进行。

1. advanced version
2. 各节点读取相应的分配elements，
3. 根据element大小，给num\_int\_transfer分配一个适当值。 （num\_int\_transfer就是相邻节点交换的elements个数）
4. 各节点内部quicksort
5. 循环直到节点0通知排序完成：
   1. 最小的num\_int\_transfer个elements发送给左边节点，

最大的num\_int\_transfer个elements发送给右边节点

* 1. 从左右节点各接收num\_int\_transfer个elements

并选择自己发送给左边节点的num\_int\_transfer个elements与从左边节点接收到的num\_int\_transfer个elements中的较大2\*num\_int\_transfer个elements，并插入原始elements队列。

同理

并选择自己发送给右边节点的num\_int\_transfer个elements与从右边节点接收到的num\_int\_transfer个elements中的较小2\*num\_int\_transfer个elements，并插入原始elements队列。（插入的同时保证elements队列有序）

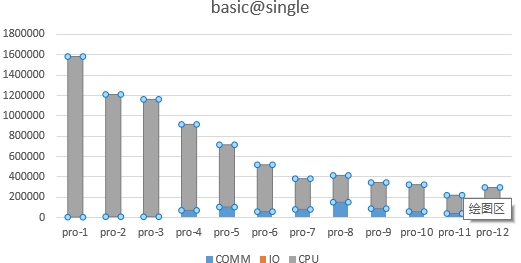
* 1. 判断是否有跟相邻节点交换elements，并通知节点0

节点0手机所有其他节点信息，并在通知其他节点是否还需要继续排队

# 实验结果

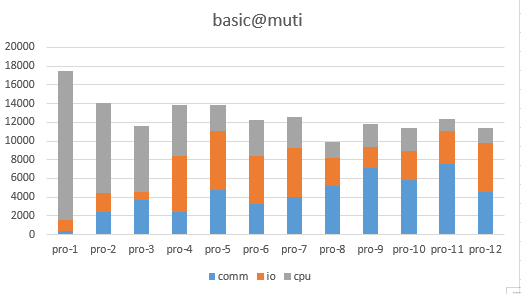
basic single node:

事实上由于processor增加IO时间也略有增加，但是主要时间还是在CPU计算上



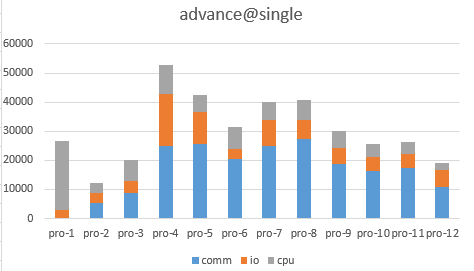
basic multiple node:

随着节点增多，IO所占时间比例变大， 同时communication的时间比例也有增大。

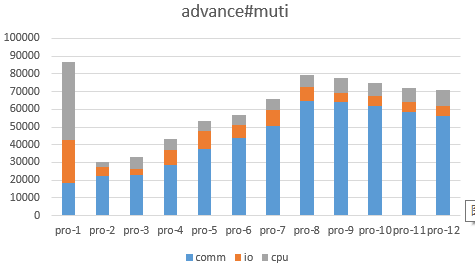


advance版本中相邻process之间传输的数据量要大，所以communication时间开销稍大

此外由于advanced版本的的时间明显低于basic版本，所以IO的开销明显增大

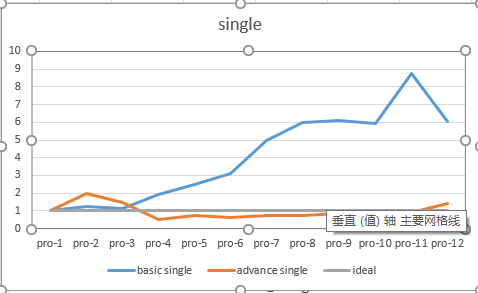


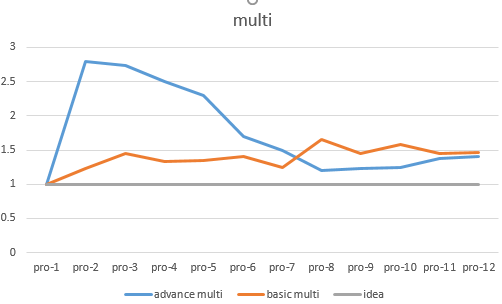
advanced multiple node版本：主要瓶颈是communication



# speedup

speed up 图表中y坐标是Ts/Tp, 没有 乘p.





# MPI IO,SEQUENCE IO

由于测试数据（1M）量比较小的原因，导致MPI IO 的时间开销明显更大，但是在process数目增大以后差距变小。

推断在数据量足够大以及process数目够多时， MPIIO效率会明显高于sequence IO



# 其他

实验中通过分奇偶的方式提高并行度

rank为奇数的节点发送时rank为偶数的节点接收，rank为偶数的节点发送时rank为奇数的节点接收。

实验中通过调整相邻节点中传输的elements数目提高效率。

这次实验，对于一个大学部的学生来说难度很大，编程中有很多的边际条件需要考虑，锻炼了编程能力； 熟悉linux命令的使用以及sh脚本的编写。

主要困难： 实验性要求高是我以前没有遇到过的。

ssh登录后，有时候会很慢很慢。