小作业二：MPI Allreduce

计04 何秉翔 2020010944

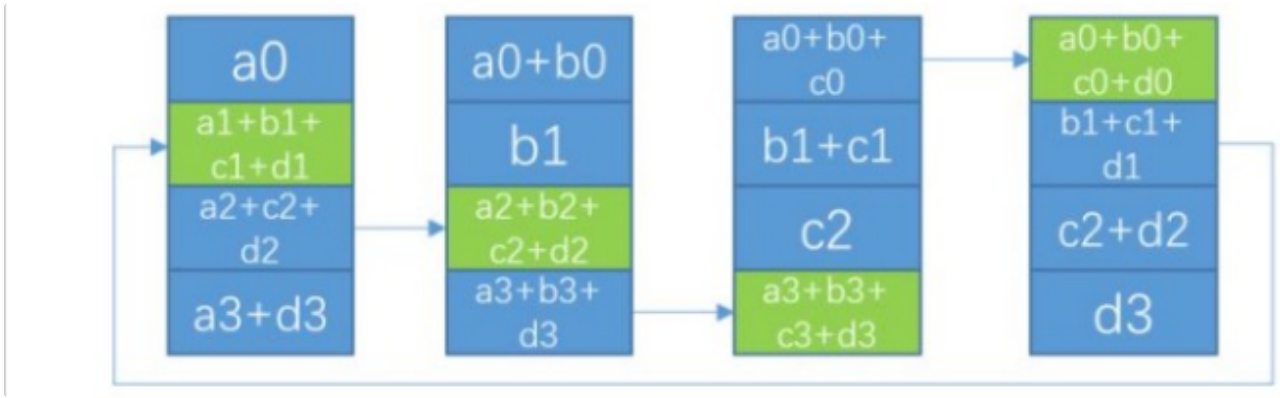
1. Ring_Allreduce 实现

我们将 `all_reduce` 的过程分成两个阶段，下面两个阶段分别介绍。

设总共的进程数为 p，当前进程为 $i \in [0, p)$，需要进行 `all_reduce` 的 `float` 个数为 n，为了防止死锁，所有的通信以**非阻塞方式**进行。

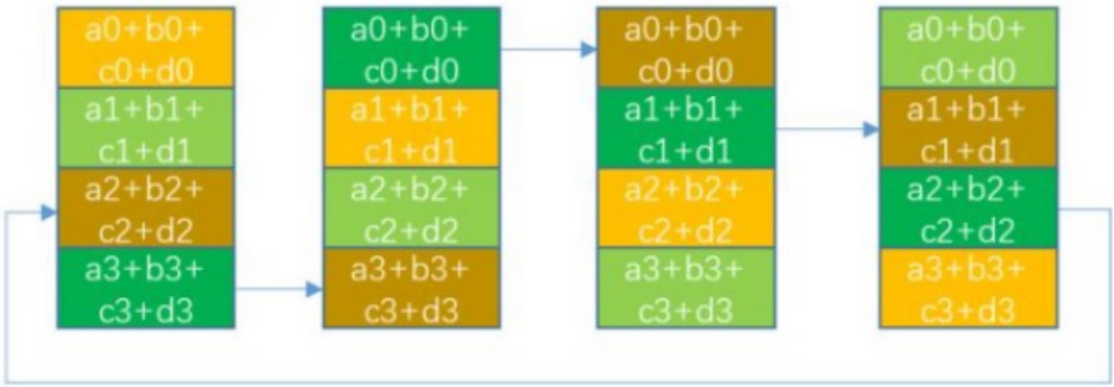
1.1 stage 1: reduce scatter

1. 首先将当前进程 i 的数据 `sendbuf` 分为 p 个数据块，数据块编号 $j \in [0, p)$，每个数据块的大小为 $(n+p-1)/p$，由于 p 不一定整除 n，所以至多最后一个数据块未填满。
2. 对于当前进程 i，接着连续进行 $p-1$ 步操作，对于第 $k \in [0, p-1)$ 步操作，做如下的事情：
   - **非阻塞**：将自己的第 $(i-k+p)\%p$ 个数据块发送给第 $(i+1)\%p$ 个进程。
   - **非阻塞**：从第 $(i-1+p)\%p$ 个进程那接受数据到一个缓冲 `buf` 中。
   - 使用 `MPI_Waitall` 等待通信完成。
   - 将接受的 `buf` 中的数据**累加**到 `sendbuf` 对应位置中，以待下一步操作。
3. 在第 $p-1$ 步结束后，第 i 个进程的第 $(i+1)\%p$ 个数据块保存了所有进程的同位置数据块的累加和，如 `ppt` 图所示：



1.2 stage 2: all gather

1. 对于当前进程 i，接着连续进行 $p-1$ 步操作，对于第 $k \in [0, p-1)$ 步操作，做如下的事情：
   - **非阻塞**：将自己的第 $(i+1-k+p)\%p$ 个数据块发送给第 $(i+1)\%p$ 个进程。
   - **非阻塞**：从第 $(i-1+p)\%p$ 个进程那接受数据到一个缓冲 `buf` 中。
   - 使用 `MPI_Waitall` 等待通信完成。
   - 将接受的 `buf` 中的数据**赋值**到 `sendbuf` 对应位置中，以待下一步操作。
2. 在第 $p-1$ 步结束后，每个进程的每个数据块都是所有进程对应位置数据块的累加和，即 `all_reduce` 目标达成，如 `ppt` 图所示：



2. 通信时间测试

在此部分，我们改变使用的节点数 $N \in \{1, 2\}$，进程数 $p \in \{2, 4, 8, 16\}$，以及需要通信的 `float` 数 $n \in \{1K, 32K, 1M\}$，测得三种 `all_reduce` 通信算法运行时间如下表所示：

进程数 p	通信量 n	节点数 N	MPI_Allreduce	Naive_Allreduce	Ring_Allreduce
2	$1K$	1	$0.115303ms$	$0.22235ms$	$0.12677ms$
2	$32K$	1	$1.41175ms$	$1.77609ms$	$1.12869ms$
2	$1M$	1	$23.1996ms$	$30.3268ms$	$21.9561ms$
4	$1K$	1	$0.187502ms$	$0.306813ms$	$0.195607ms$
4	$32K$	1	$2.18847ms$	$3.22673ms$	$1.90558ms$
4	$1M$	1	$32.9582ms$	$46.5607ms$	$29.8168ms$
8	$1K$	1	$0.233642ms$	$0.31693ms$	$0.224757ms$
8	$32K$	1	$2.0692ms$	$3.38255ms$	$2.62621ms$
8	$1M$	1	$46.7107ms$	$57.877ms$	$43.2431ms$
16	$1K$	1	$0.328218ms$	$0.372476ms$	$0.335371ms$
16	$32K$	1	$1.67368ms$	$3.23829ms$	$1.51073ms$
16	$1M$	1	$53.1162ms$	$66.7656ms$	$53.4931ms$
2	$1K$	2	$0.097643ms$	$0.119063ms$	$0.084409ms$
2	$32K$	2	$0.909532ms$	$1.25772ms$	$0.864494ms$
2	$1M$	2	$14.9008ms$	$22.3426ms$	$14.0672ms$
4	$1K$	2	$0.201034ms$	$0.291444ms$	$0.214173ms$
4	$32K$	2	$2.37275ms$	$3.86773ms$	$2.29529ms$
4	$1M$	2	$31.4416ms$	$45.6233ms$	$26.902ms$
8	$1K$	2	$0.322703ms$	$0.415348ms$	$0.32716ms$
8	$32K$	2	$2.75309ms$	$5.24294ms$	$3.23659ms$
8	$1M$	2	$39.0054ms$	$59.1673ms$	$41.8886ms$
16	$1K$	2	$0.507563ms$	$0.515228ms$	$0.491788ms$
16	$32K$	2	$2.33681ms$	$4.41913ms$	$1.95505ms$
16	$1M$	2	$48.6776ms$	$77.7277ms$	$51.8548ms$

可以看出：

1. 相同节点数和通信量时，运行时间基本随着进程数增加而增加，这可能由于实现中没有考虑计算和通信重叠，而是直接在非阻塞通信后进行 `MPI_Waitall`，因此进程数越多通信时间占比越大
2. 相同进程数和节点数时，运行时间随着通信量增加而增加。
3. 相同进程数和通信量时，节点间进程通信和节点内进程通信性能不同，没有太明显的对比。