Parallel Sorting

1 Overview

In this chapter we will visit the problem of parallel comparison-based sorting on distributed memory machines. The input is a distributed representation of an array $A[0 \dots n-1]$ containing n elements and a comparison function to compare any two elements. The output is a distributed representation of the sorted array $B[0 \dots n-1]$. If p denotes the number of processors, we will assume $n >> p$. For ease of exposition, we will assume n to be divisble by p.

Pre-condition: Every processor stores $O(\frac{n}{p})$ distinct elements of the unsorted array A initially. We will denote the array local at rank i by A_i.

Post-condition: The sorted array $B[0 \dots n-1]$ is stored in a distributed manner, such that rank i stores the i^{th} $O(\frac{n}{p})$ segment of the sorted array. We will denote the portion of the sorted array output at rank i by B_i.

In this chapter, we will describe two parallel sorting algorithms.

1.1 Sample sort

Sample sort is an extension of the partitioning-based scheme used in conventional quick sort. The main idea is to identify pivots to partition the input array so that the individual parts can then be subsequently sorted locally within each processor. Finding pivots that can evenly partition the input array into roughly p equal parts can be done in many ways; however, size bounds (on the output array) are harder to guarantee. For instance one can collect statistics such as minimum and maximum to decode the range but a skewed input could cause potential imbalances among processor workloads.

Traditional sample sort-based methods adopt typically a two-step approach: of first generating a set of "local pivots" based on the local distribution and using those pivots to generate a set of "global pivots" to perform the final partitioning.

The algorithmic template for sample sort is as follows:

S1) Every processor sorts its local array A_i, and picks $p-1$ evenly spaced pivots from the local sorted array. These local pivots are also referred to as "*local splitters*". We will also refer to the sorted array as A_i^s, although A_i^s could be the same array A_i if sorting is done inplace.

S2) The $p(p-1)$ splitters generated across all p processors are sorted, and subsequently a subset of $p-1$ evenly spaced pivots are selected. These pivots are also referred to as the "*global splitters*". The sorting in this step can be either performed in parallel (using another parallel sorting method such as bitonic sort) or in serial, depending on the size of p. For the purpose of analysis in this chapter, we will assume this step is performed using first an Allgather communication of the local splitters, followed by each processor locally sorting all $p(p-1)$ pivots to obtain the $p-1$ global splitters.

S3) Using the global splitters, each processor partitions its local array into p (possibly empty) parts, marking the i^{th} part to be sent to processor rank i.

S4) The arrays are redistributed using the Alltoallv transportation primitive.

S5) The set of elements received at every processor is then locally sorted. This can either be achieved using a local sort of all the elements achieved, or as a local merge operation of p sorted lists. Note that the individual parts received from each source processor in the Alltoallv step are already sorted internally.
At the end of this step, the concatentation of the local arrays (B_i) from rank 0 to rank $p-1$ in that order represents the final sorted output (B).

Let n_i be the number of elements in the sorted array output by rank i at the end of sample sort. In the above version of sample sort, this number (which denotes the output size within each processor) can be bounded.

Lemma 1.1. *The number of elements in the sorted array output by any rank i at the end of sample sort is $O(\frac{n}{p})$.*

Proof. In the sample sort algorithm let the sequence of $(p-1)$ global splitters be labelled $\{s_0, s_1, \ldots, s_{p-2}\}$. Processor p_i ($i > 0$) receives all elements x in the input array A s.t. $s_{i-1} < x \leq s_i$, while processor p_0 receives all $x \leq s_0$. Let us refer to the set of values received at a given processor p_i as $Range(i)$. Note that we are interested in determining an upper-bound for the size of this $Range(i)$. In what follows, we show that $|Range(i)| < 2 \times \frac{n}{p}$.

It should be easy to see that this set $Range(i)$ could include a subset of local splitters that are generated in different processors. In fact there has to be exactly $(p-1)$ local splitters included in the $Range(i)$. This is because the global splitters themselves were originally picked by selecting $(p-1)$ evenly spaced splitters from the sorted array of $p(p-1)$ local splitters (step S2).

These $(p-1)$ local splitters can originate from one or more processors. This gives raise to two sub-cases:
Case A) All $(p-1)$ local splitters originate from the same processor.
Case B) The $(p-1)$ local splitters originate from two or more processors.

Case A) Let the $(p-1)$ local splitters originate from processor p_j. This implies a possibility that all the $\frac{n}{p}$ elements of the local input array $A_j$could belong to $Range(i)$. Now, what can we say about the contribution of other processors p_k ($k \neq j$) to p_i? We show here that the net contribution from these remaining processors cannot exceed $\frac{n}{p}$. The proof is as follows: Since p_j contributed all the $(p-1)$ local splitters in $Range(i)$, other processors could have not contributed any local splitters. However there could still be elements in their respective local input arrays that belong to $Range(i)$. If there are such elements, they need to be occurring prior to the first local splitter or after the last local splitter within those processor's sorted arrays (i.e., A_k^s). This implies that the maximum contribution from any such p_k to $Range(i)$ is bounded by $\frac{n}{p^2}$. Therefore, the net contribution to $Range(i)$ from any p_k ($k \neq j$) can be bounded by $(p-1) \times \frac{n}{p^2} < \frac{n}{p}$. Therefore, $|Range(i)| < 2 \times \frac{n}{p}$.

Case B) Let k_j denote the number of local splitters contributed by processor p_j to $Range(i)$. If $k_j > 1$ then all the elements in A_j^s between those local splitters are also in $Range(i)$. Note that there are exactly $\frac{n}{p^2}$ elements between any two consecutive local splitters of A_j^s. In addition, it is possible that a subset of the $\frac{n}{p^2}$ elements preceding the first of those k_j splitters or following the last of those k_j splitters (but not both) could also belong in $Range(i)$. This implies that the maximum contribution from p_j to $Range(i)$ is $(k_j + 1)\frac{n}{p^2}$. Therefore, the total contribution from all p processors to $Range(i)$ is given by:

$$\begin{aligned} |Range(i)| &\leq \Sigma_{j=0}^{p-1}(k_j+1)\frac{n}{p^2} \\ &= \frac{n}{p^2}(\Sigma_{j=0}^{p-1}(k_j) + \Sigma_{j=0}^{p-1}1) \\ &= \frac{n}{p^2}(p-1+p) \\ &< 2 \times \frac{n}{p} \end{aligned} \tag{1}$$

□

Analysis: The sample sort algorithm has the following complexity:

Step	Computation time	Communication time
S1	$\Theta(\frac{n}{p}\lg\frac{n}{p})$	
S2[1]	$\Theta(p^2\lg p^2)$	$O(\tau\lg p + \mu p^2)$
S3	$O(\frac{n}{p})$	
S4		$O(\tau p + \mu\frac{n}{p})$ $(\because |Range(i)| < \frac{n}{p})$
S5	$\Theta(\frac{n}{p}\lg\frac{n}{p})$	
Total:	$O(\frac{n}{p}\lg\frac{n}{p} + p^2\lg p^2)$	$O(\tau p + \mu(p^2 + \frac{n}{p}))$

1.2 Bitonic sort

Please refer to the notes by Prof. Aluru for a detailed discussion on Bitonic sort.

[1]Note that the quadratic terms in step S2 can be brought down by performing that sort in parallel.