Optimizing Sparse Vector MPI Reduction Operations

ABSTRACT

We describe optimizations to MPI reduction operations when the input data vectors have higher fraction of neutral elements for a given binary operator. In the target problem, termed as *sparse reduction*, each process of a set of processes contribute input data vectors with different sizes by ignoring neutral elements. Sparse reduction is considerably more general than the problems that can be solved with MPI collective reduction operations, where all processes contribute data vectors with the same number of elements. We give optimal constructions of binary reduction trees for sparse parallel reduction problem with associative operators. An efficient selection of reduction algorithms is also presented which is based on varying sparsity, distribution of non neutral elements in the input vectors, number of processes and the communication size. Finally, using proposed sparse reduce optimizations, we give indications of practical savings in run time of a few machine learning applications, as compared to MPI_Reduce.

1 INTRODUCTION

2 BACKGROUND AND MOTIVATION

3 OPTIMAL BINARY TREES

4 SPARSE REDUCE (Sreduce)

In this section, we propose optimizations to MPI Reduce based on sparsity and distribution of non-neutral elements. Table 1 describe important notations used in the algorithm. Algorithm 1: Sreduce, requires four input parameters and threshold variables in Table 1, to select a reduction structure. It is assumed that the data is non-sparse in case value for sparsity (S) is unknown. Where as, D is assumed to be $uniform$ by default. [Jagpreet: experimentally find threshold values]

4.1 BINOMIAL REDUCTION TREE

We use binomial tree reduction structure with a small variation of using an optional pipeline which is decided locally by each process participating in the collective. Initially, the input vectors are sparse and medium in size. The input vectors are classified as medium if in the dense representation of the vector, binomial tree is not an efficient reduction structure. Dependent on the distribution of non-neutral elements, the intermediate resultant vectors at various processes may become dense. Hence, for these vectors, processes switch to the dense representation. These medium size dense vectors are divided into smaller sizes and pipelined to parent process for efficient transfer. [Jagpreet: test local pipeline]

To efficiently handle both uniform and zipfian distributions, we use two different thresholds on number of processes: T_u^p and T_z^p respectively. Where, $T_z^p > T_u^p$, because in uniform distribution of non-neutral elements, the intermediate vectors may become dense far more quickly than the zipfian distribution.

Table 1: Notations

P	Number of processes
C	Size of the dense vector
S	Sparsity of input vectors i.e., fraction of neutral elements
D	Distribution of non-neutral elements ($uniform$ or $zipf$)
T^c, T_s^c	$T^c < T_s^c$, threshold on C, independent and dependent of S respectively
T^s	Threshold on sparsity
T_u^p, T_z^p	$T_u^p < T_z^p$, threshold on number of processes when distribution of non-neutral elements is uniform or zipfian respectively

Algorithm 1: Sreduce

```
Data: P: # processes, C:dense vector size, S:sparsity,
      D:Distribution
if (C < T^c) then
    /* small communication size                                   */
    call Binomial_LocalPipe_Reduce;
else if C < T_s^c and S > T^s and P < T_z^p and D = zipf then
    /* medium communication size, sparse data, Zipf distribution  */
    call Binomial_LocalPipe_Reduce;
else if C < T_s^c and S > T^s and P < T_u^p and D = uniform then
    /* medium communication size, sparse data, uniform distribution
       */
    call Binomial_LocalPipe_Reduce;
else
    /* large communication size, dense data, call binary or linear
       pipeline based on size                                     */
    call Binary2tree_Pipe_reduce;
end
```


© 2017 ACM. ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235