Scalability of Hybrid Sparse Matrix Dense Vector (SpMV) Multiplication

Brian A. Page
CSE Dept.
Univ. of Notre Dame
Notre Dame, IN USA
bpage1@nd.edu

***Abstract*—Reproduce a Sparse Matrix Dense Vector Multiplication (SpMV) experiment in order to increase the cluster size and thread count for the purpose of analyzing scalability as these increase. We will examine several different work distribution methods (overflow, optimal, and "splitMatrix") in order to evaluate how these distribution methods behave as we scale the problem size as well as cluster and thread size. We will be utilizing the message passing interface (MPI) for communication between distributed memory segments (cluster nodes) as well as OpenMP to enable shared memory segments locally on each cluster node.**

***Keywords*-Scalability, Hybrid SpMV, Knights Landing;**

I. Introduction

II. Related Work

III. Load Balancing Implementation

A. Overflow

B. Split Matrix

bla bla blah [1]

To Do: update the following pseudo-code to represent the Split Matrix algorithm

Algorithm 1 Distributed Sub Matrix

```
1: procedure SPLITMATRIX
2:     Input: matrixCSR A[3], distribution[p][3][k]
3:     for i ← 0 − matrixCSR[i].size() do
4:         for j ← matrixCSR[i][1][0] − do
5:         if rowLengths_i.size() > nBF then
6:             splitRow(i,rowLengths)
```

C. Optimal

In order to generate a work distribution with the highest balance, a greedy bin packing algorithm was utilized. A greedy algorithm was chosen for its simplicity of design and implementation. Once the sparse matrix being worked on has been converted into CSR format it can be segmented into k portions, where k is the number of nodes in the computing cluster.

The end outcome of computation is the dense vector resulting from the SpMV operation, therefore each element present in the CSR representation of the sparse matrix A, must be multiplied by the correct element in the dense vector x, to produce b. If a row or column is spread across multiple nodes it is necessary to send the corresponding element in x to each cluster node to insure proper calculation. Similarly additional reductions may be required in order to compile the final value for the given column or row. This creates additional overhead that exists due to the extra data that must be distributed amongst the cluster.

To aid in the reduction of the aforementioned overhead our greedy bin packing algorithm seeks to maintain rows or columns in their entirety on a single node. The entire i^{th} row/column may then be multiplied with a single element from the dense vector. Additionally only a single gather must be performed in order for the master to obtain the value b_i. There will however be cases in which it may prove impossible to avoid inordinately large rows/columns present within the matrix. In such cases we would like to see such a row/column distributed to the least number nodes possible.

The $node\ balance\ factor$, the optimal number of elements each cluster node should be given to work on, is determined from information known about the matrix and cluster size.

$$nodeBalanceFactor = \frac{nonZeros}{proc}$$

Once calculated $node\ balance\ factor$ is used as the metric for determining if a row should be split and sent to multiple nodes. A row split of this nature is necessary as the optimal work distribution mandates that the number of elements distributed to each node be as close to the $node\ balance\ factor$ as possible, therefore should a single row exceed this amount it must be split in order to achieve optimal distribution across all cluster nodes.

Should a row/col require splitting, it is split into the necessary number of sub rows with each sub row being placed into the row vector and treated the same as all other rows for the purpose of load balancing. These sub rows share the same row id but contain the elements unique to their portion of the original large row. After sorting the vector of rows by row size, we iterate over the sorted vector from largest to smallest row, assigning the i^{th} row to the cluster nodes with the lowest number of assigned elements. If two or more nodes have the current minimum assignment, the

i^{th} is assigned arbitrarily to one node in the minimum set.

Upon balancing, the row-to-node assignments are used to control work load packing which will take all rows assigned to a given node and generate that node's private CSR representation of the data it shall work on. This was done in order to reduce the number of transfers between the master MPI process and slave processes, as well as to increase the locality of reference for accessing the data to be sent.

So that calculation is carried out correctly, each node must be sent the elements of the dense vector that correspond to the rows assigned to that node. We take advantage of the row-to-node assignment structure which contains the row ids and the node to which they have been assigned. In this it is possible to compile a private dense vector for each cluster node such that

$$nodeResult_i = row_i * denseVec_i$$

Upon receipt of data each node proceeds with computation without regard to the actual rows it may be working on thanks to the prior knowledge contained within the node assignment structure.

Algorithm 2 Greedy Balanced Distribution

```
1:  procedure GREEDYPACK
2:      Input: matrixCSR A[3], distribution[p][4][k]
3:      nBF ← nonZeros / processCount
4:      for i ← 0 − n do
5:          if rowLengths_i.size() > nBF then
6:              splitRow(i,rowLengths)
7:      Sort(rowLengths)
8:      for i ← 0 − rowLengths.size() do
9:          node ← 0
10:         min ← distribution[node][0][0]
11:         for j ← 0 − processCount do
12:             if distribution[j][0][0] < min then
13:                 node ← j
14:                 min ← distribution[j][0][0]
15:         distribution[node] ← rowLengths_i
```

IV. Evaluation

A. Benchmarks

B. Impact: Load Balancing

C. Impact: Sparsity

V. Future Work

VI. Conclusions

Acknowledgment

References

[1] B. Bylina, J. Bylina, P. Stpiczyski, and D. Szakowski, "Performance analysis of multicore and multinodal implementation of spmv operation," *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, vol. 2, pp. 569–576, Oct 2014.