Basic Concepts for Distributed Sparse Linear Algebra Operations *

Victor Eijkhout and Roldan Pozo

University of Tennessee
Department of Computer Science
107 Ayres Hall, Knoxville, TN 37996-1301
eijkhout@cs.utk.edu, pozo@cs.utk.edu

August 4, 1994

1 Introduction

We introduce basic concepts for describing the communication patterns in common operations such as the matrix times vector and matrix transpose times vector product, where the matrix is sparse and stored on distributed processors. At first we will describe a simple one-dimensional partitioning of the matrix, then we will describe the more general case where arbitrary elements are assigned to processors.

2 One-dimensional matrix partitioning

We start by describing a one-dimensional partitioning of the matrix, that is, a distribution of the matrix rows or columns to the processors. The discussion will describe only the distribution by rows, but the translation to a column partitioning is easily made.

We assume that there exists a map

$$\text{map}: \mathbf{N} = \{1, \ldots, N\} \to \mathbf{P} = \{1, \ldots, P\}$$

where N is the number of problem variables, and P is the number of processors. From this map we construct the sets V_p of the 'owned variables' of the processors:

$$V_p = \{i: \text{map}(i) = p\}. \tag{1}$$

It is clear that

$$V_p \subset \mathbf{N}, \quad \bigcup_p V_p = \mathbf{N}, \quad V_p \cap V_q = \emptyset \text{ if } p \neq q.$$

*. This work was supported by DARPA under contract number DAAL03-91-C-0047

We denote the number of owned variables of a processor by

$$\mathbf{N}_p = \{1, \ldots, N_p\}, \qquad N_p = |V_p|.$$

The focus of the discussion here is on operations such as the matrix vector product $y = Ax$. Suppose that initially the matrix and vectors are distributed along the map() function, then for $i \in V_p$

$$y_i = \sum_j A_{ij} x_j = \sum_{j \in V_p} A_{ij} x_j + \sum_{j \notin V_p} A_{ij} x_j .$$

Clearly, some communication between processor p and other processors will be needed. We will give some definitions to make the communication pattern more precise.

For this we define two other types of sets of variables. First of all, the 'border variables' of a processor are those variables owned by other processors that border on this processor:

$$B_p^q = \{j \in V_q : \exists_{i \in V_p} A_{ij} \neq 0\} \tag{2}$$

Next, the 'edge variables' of a processor are those variables owned by that processor that border on other processors:

$$E_p^q = \{j \in V_p : \exists_{i \in V_q} A_{ij} \neq 0\} \tag{3}$$

Occasionally we will refer to

$$B_p = \bigcup_q B_p^q, \qquad E_p = \bigcup_q E_p^q .$$

From the definitions it is clear that $B_q^p = E_p^q$.

3 Matrix vector product under one-dimensional decomposition

The usefulness of the concepts of border and edge variables becomes apparent when we consider the matrix and matrix transpose times vector products $y = Ax$ and $y = A^t x$ with matrices partitioned over the processors by rows or columns.

For $i \in V_p$

$$y_i = \sum_j A_{ij} x_j = \sum_{j \in V_p} A_{ij} x_j + \sum_{q \neq p} \sum_{j \in B_p^q} A_{ij} x_j . \tag{4}$$

With a block partitioning along processor variables, e.g., $y_{(p)}$ is the subvector of y containing only the values in V_p, and a corresponding two-dimensional block partitioning of the matrix, we can write this shorter as

$$y_{(p)} = A_{(pp)} x_{(p)} + \sum_{q \neq p} A_{(pq)} x_{(q)} .$$

Note that $A_{(pq)}$ is non-null iff B_p^q is non-empty.

Now, for a practical implementation of the above formula we consider a matrix data distribution where processor p has the matrix block row $A_{(p*)}$. If additionally it has the input vector x, it can compute the values $y_{(p)}$ in its owned variables. In practice, the matrix will be sparse, and a processor need not have the whole vector x, merely those blocks $x_{(q)}$ for which $B_p^q \neq \emptyset$.

We will call $V_p \cup \bigcup_q B_p^q$ the 'local variables' of a processor. A processor then needs the values of x exactly in its local variables in order to compute the values of y in its owned variables. Under the assumption that initially the values of x reside only in the owned variables, some amount of communication is needed.

In terms of the border and edge sets, computing a matrix vector product as in formula (4) entails the following actions for a processor p, though not necessarily in this sequence:

- for each q such that $E_p^q \neq \emptyset$, send $x_{(q)}$ to processor q.
- for each q such that $B_p^q \neq \emptyset$, receive $x_{(q)}$ from processor q.
- compute the partial result $A_{(pp)}x_{(p)}$ acting on x in the owned variables.
- compute the partial results $A_{(pq)}x_{(q)}$ acting on x in border variables.
- sum all partial results to form $y_{(p)}$ in the owned variables.

For a matrix partitioned by block rows, the transpose matrix vector product is somewhat more tricky. For $y = A^t x$ we find for $i \in V_p$

$$y_i = \sum_j (A^t)_{ij} x_j = \sum_j A_{ji} x_j = \sum_{j \in V_p} A_{ji} x_j + \sum_{q \neq p} \sum_{j \in E_p^q} A_{ji} x_j , \tag{5}$$

or again shorter:

$$y_{(p)} = A^t_{(pp)} x_{(p)} + \sum_{q \neq p} A^t_{(qp)} x_{(q)} .$$

Under the above assumption that processor p is in possession of the full block row $A_{(p*)}$, this expression is no longer computable, since it involves the block column $A_{(*p)}$. Hence we arrive at a scheme where processor q computes the partial results $y^{(q)}_{(p)} = A^t_{(qp)} x_{(q)}$ (for all $p \neq q$ for which $A_{(qp)} \neq 0$), and sends it to processor p. Processor p then constructs

$$y_{(p)} = A^t_{(pp)} x_{(p)} + \sum_q y^{(q)}_{(p)} .$$

In terms of the border and edge sets, computing the matrix transpose vector product as in formula (5) then entails the following actions for a processor p:

- compute the partial result $y^{(p)}_{(p)}$ in the owned variables as $A^t_{(pp)} x_{(p)}$.
- for each q such that $B_p^q \neq \emptyset$, compute the partial results $y^{(p)}_{(q)}$ in the border variables as $A^t_{(pq)} x_{(p)}$.
- for each q such that $B_p^q \neq \emptyset$, send $y^{(p)}_{(q)}$ to processor q.
- receive $y^{(q)}_{(p)}$ from processor q.
- sum all partial results to form $y_{(p)}$ in the owned variables.

In the case of a matrix partitioned into block columns the above transpose product algorithm is used for the regular product and vice versa. A symmetrically stored matrix can be considered the sum of an upper triangular matrix stored by rows plus a lower triangular matrix stored by columns, so a hybrid of the above algorithms applies. Below we will describe a further partitioning of the block rows or columns, that is, an arbitrary assignment of matrix elements to processors.

4 Arbitrary partitioning

In certain applications it may make sense to partition sparse matrix elements in an arbitrary manner over the processors. This generalizes the above partitioning by allowing for instance matrix blocks to be assigned to a processor. In particular, in this manner a processor need not posess any diagonal elements of the matrix. We will describe a fully general assignment of matrix elements to processors, extend the definitions of border and edge sets to this case, and outline the matrix times vector product under such an assignment scheme.

We formulate this case for a general rectangular matrix of size $N_1 \times N_2$. We will assume a set of processors, and a mapping of coordinates (i, j) in the matrix to the processor set:

$$\text{map} : \mathbf{N}_1 \times \mathbf{N}_2 = \{1, \ldots, N_1\} \times \{1, \ldots, N_2\} \to \mathbf{P} = \{1, \ldots, P\}.$$

The partitioning sets induced by this mapping

$$V_1(p) = \{i : \exists_j \text{map}(i, j) = p\}, \qquad V_2(p) = \{j : \exists_i \text{map}(i, j) = p\}$$

form, not necessarily disjoint, splittings of $\mathbf{N}_1$ and $\mathbf{N}_2$.

The case of one-dimensional partitioning by rows is obtained by choosing a mapping function such that

$$V_2 \equiv \mathbf{N}_2$$

and such that the V_1 sets be a disjoint splitting of N_1.

Corresponding to the V_1 and V_2 sets we have two block partitionings of vectors, and one of the matrix. If $x \in \mathbf{R}^{N_1}$, $x_{(p;1)}$ is the vector defined by

$$\left(x_{(p;1)}\right)_i = \begin{cases} x_i & \text{if } i \in V_1(p) \\ 0 & \text{otherwise} \end{cases}$$

and similarly we define $x_{(p;2)}$ using the $V_2(p)$ sets. The matrix is partitioned along the processor map: $A_{(p)}$ is the matrix for which

$$\left(A_{(p)}\right)_{ij} = \begin{cases} A_{ij} & \text{if map}(i, j) = p \\ 0 & \text{otherwise} \end{cases}.$$

The practical interpretation of all this is that processor p owns the matrix subblock $A_{(p)}$. In the context of computing $y = Ax$, processor p then needs $x_{(p;2)}$ with which it can compute $A_{(p)}x_{(p;2)}$, which is part of $y_{(p;1)}$.

Next we have to determine where processors get their input for the matrix vector product, and where they deposit their output. This is described by two ownership functions, c_1 and c_2, the first describing ownership of the input, the second of the output. Formally, they are functions

$$c_1 : \mathbf{N}_1 \to \mathbf{P}, \qquad c_2 : \mathbf{N}_2 \to \mathbf{P},$$

inducing disjoint partitionings of $\mathbf{N}_1$ and $\mathbf{N}_2$.

In the case of a one-dimensional partitioning of a square matrix, we have

$$c_1 \equiv c_2, \qquad c_1(i) = p \text{ if map}(i, i) = p,$$

that is, a variable is owned by a processor if the diagonal element in that row is owned by that processor.

The ownership functions are now used to define B and E subsets of both $\mathbf{N}_1$ and $\mathbf{N}_2$. The subsets of $\mathbf{N}_2$ will be the familiar border and edge sets, that is, the sets of variables touched but not owned, and owned and exported respectively. Additionally, we will now have subsets of $\mathbf{N}_1$ corresponding to computed but not owned, and owned but not computed variables.

Formally,

$$B_p^{(2)} = \{j \in V_2(p) \colon c_2(j) \neq p\},$$

and, split by surrounding processors,

$$B_p^{q^{(2)}} = \{j \in V_2(p) \colon c_2(j) = q\}.$$

With these definitions we have

$$\bigcup_{q \neq p} B_p^{q^{(2)}} = B_p^{(2)}, \qquad B_p^{(2)} \cup B_p^{p^{(2)}} = V_2(p).$$

We define

$$E_p^{q^{(2)}} = B_q^{p^{(2)}} = \{j \in V_2(q) \colon c_2(j) = p\}.$$

These definitions coincide with the definitions for B_p, B_p^q, E_p, and E_p^q, in the one-dimensional case, with the reduction described as above.

Similarly, we define

$$B_p^{(1)} = \{j \in V_1(p) \colon c_1(j) \neq p\},$$

and, split by surrounding processors,

$$B_p^{q^{(1)}} = \{j \in V_1(p) \colon c_1(j) = q\}.$$

With these definitions we have

$$\bigcup_{q \neq p} B_p^{q^{(1)}} = B_p^{(1)}, \qquad B_p^{(1)} \cup B_p^{p^{(1)}} = V_1(p).$$

Also, we define

$$E_p^{q^{(1)}} = B_q^{p^{(1)}} = \{j \in V_1(q) \colon c_1(j) = p\}.$$

5 Matrix vector product under arbitrary decomposition

The above definitions are now employed to describe the matrix vector product for a matrix with arbitrary assignment of elements to processors.

1. Every processor p has to gather the elements of $x_{(p;2)}$, and help other processors in constructing their parts of x:
   - for each q for which $E_p^{q^{(2)}}$ is not empty, send the components of x in this set;
   - for each q for which $B_p^{q^{(2)}}$ is not empty, receive the components of x in this set.
2. Compute $y_{(p;1)}^{(p)} := A_{(p)} x_{(p;2)}$.
3. Every processor p now has a part of $y_{(p;1)}$, and it has to distribute components of that to whoever owns those:

- for each q for which $B_p^{q^{(1)}}$ is not empty, send the components of $y_{(p;1)}^{(p)}$ in this set;
- for each q for which $E_p^{q^{(1)}}$ is not empty, receive the components of x in this set, and add them to what is already stored in this component.

Note that this algorithm generalizes the matrix and transpose matrix vector product algorithms of section 3: for the regular product only steps 1 and 2 are needed then, while the transpose product uses steps 2 and 3.

6 Conclusion

We have introduced basic concepts for describing communication in common linear algebra operations on matrices stored on distributed processors. For the case of a one-dimensional decomposition of the matrix the concepts of owned, local, border, and edge variables have, although defined strictly in terms of the matrix sparsity pattern, an easy physical interpretation in terms of proximity of problem variables. The generalization of these concepts to arbitrary matrix partitionings has no such interpretation. In fact, we need to define the concepts of border and edge for both input and output in the matrix vector product.

In a companion paper, [1], we will discuss data structures and algorithms for specific realizations of the theoretical concepts introduced here.

References

[1] Victor Eijkhout and Roldan Pozo. Data structures and algorithms for distributed sparse matrix operations. Technical Report in preparation, Computer Science Department, University of Tennessee, Knoxville, 1994.