Lab 14: Parallel sparse matrix-vector multiplication with MPI

Oleg Batrashev
version 0.3
December 11, 2009

1 Assignment: parallel sparse $Ax$

1.1 Introduction

1.1.1 Basic ideas

Let's take an example of matrix $A$ and vector $x$

\[ y = Ax = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 2 & 0 & 2 \\ 0 & -2 & 4 & -0.5 \\ 1 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1.2 \\ -3 \\ 0.22 \\ 0.2 \end{pmatrix} = \begin{pmatrix} -4.8 \\ -2 \\ 6.78 \\ 1.6 \end{pmatrix} \]

Performing sparse matrix-vector multiplication in parallel is often done by assigning each process to calculate its own part of the resulting vector $y$. How to divide $y$ into parts for efficient matrix-vector multiplication is the question, that depends on the type of initial problem, where matrix $A$ comes from. We will see how to efficiently distribute $y$ for FDM in the assignment 2. For now let's assume we have 2 processes and the process number 0 calculates $y_0$ and $y_1$, whereas the process number 1 – $y_2$ and $y_3$.

In many real-life problems it is often reasonable to distribute matrix $A$ rows and vector $x$ values between the processes in the same manner as vector $y$. This means that vector values $x_0$, $x_1$ go to process 0 and values $x_2$, $x_3$ to process number 1. Then matrix $A$ rows 0,1 should go to process 0 and rows 2,3 to process 1 (blue is for process 0, red for process 1)

\[ Ax = A \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 2 & 0 & 2 \\ 0 & -2 & 4 & -0.5 \\ 1 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1.2 \\ -3 \\ 0.22 \\ 0.2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \]

If we rewrite the calculation for each process we get

\[
\begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}
\]

\[
\begin{pmatrix} 0 & -2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} 4 & -0.5 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \end{pmatrix}
\]
which shows that the result is the sum of two multiplications: one term only uses local values and the other requires remote values of $x$. The submatrices from the latter term are local, but they need the values of $x$ from remote processes and we will refer to them as *ghost* matrices and vectors.

\[
A_{\text{local}}^{(\text{process}=0)} = \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix}, \quad A_{\text{local}}^{(\text{process}=1)} = \begin{pmatrix} 4 & -0.5 \\ 0 & 2 \end{pmatrix}
\]

\[
A_{\text{ghost}}^{(\text{process}=0)} = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix}, \quad A_{\text{ghost}}^{(\text{process}=1)} = \begin{pmatrix} 0 & -2 \\ 1 & 0 \end{pmatrix}
\]

\[
x_{\text{local}}^{(\text{process}=0)} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}, \quad x_{\text{local}}^{(\text{process}=1)} = \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}
\]

\[
x_{\text{ghost}}^{(\text{process}=0)} = \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}, \quad x_{\text{ghost}}^{(\text{process}=1)} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}
\]

The multiplication on each process then looks like the following:

1. Multiply local matrix with local $x$ values $A_{\text{local}}x_{\text{local}}$
2. Receive remote $x$ values from other processes and save them to the vector $x_{\text{ghost}}$
3. Multiply ghost matrix with received $x$ values and add up to local, so the resulting $y_{\text{local}} = A_{\text{local}}x_{\text{local}} + A_{\text{ghost}}x_{\text{ghost}}$

Our matrix is sparse we usually do not need to save all the remote $x$ values in $x_{\text{ghost}}$, but only (small) part of it. For example, there is no need to send $x_2$ from process 1 to process 0, because all values in the corresponding column of $A_{\text{ghost}}^{(\text{process}=0)}$ are zero.

1.1.2 General description

Let $d : \mathcal{N} \to \mathcal{N}$ defines distribution of vectors $x$, $y$ and rows of $A$ between processes, so that

\[
d(i) = p \implies x_i \text{ is assigned to } x_{\text{local}}^{(\text{process}=p)} \quad y_i \text{ is assigned to } y_{\text{local}}^{(\text{process}=p)}
\]

row $A_{i,*}$ stored to $A_{\text{local}}^{(\text{process}=p)}$ and $A_{\text{ghost}}^{(\text{process}=p)}$

Consider the case given on Figure 1, on process $p$:

- local vector $x_{\text{local}}^{(\text{process}=p)}$ are those $x_i$, where $d(i) = p$
- local matrix $A_{\text{local}}^{(\text{process}=p)}$ consists of all $A_{i,j}$, where $d(i) = p \land d(j) = p$
- ghost vector $x_{\text{ghost}}^{(\text{process}=p)}$ which values need to be received from other processes are all $x_j$, where

\[
\exists A_{i,j} \neq 0, d(i) = p \land d(j) \neq p
\]

\(^{1}\) *ghost* comes from the fact that they need the values of $x$ not present (ghost) on the current process, but need to update them from other processes
\[ y = AX \]
\[ A_{local} \quad x_{local} \]
\[ A^{(process=p)}_{ghost} \quad x^{(process=p)}_{ghost} \]

- in particular, values need to be received from a process \( q \) are all \( x_j \), where

\[ \exists A_{i,j} \neq 0, \; d(i) = p \land d(j) = q \]

- ghost matrix consists of all \( A_{i,j} \neq 0 \), where \( d(i) = p \land d(j) \neq p \)

Notice, that we are working with sparse matrices, i.e. many of \( A \) values are zeros, so number of values that need to be received from other processes to \( x_{ghost} \) may be quite small.

### 1.2 Implementation

All parallel code is in `parallel.py` Python module, which defines several functions to do parallel vector operations (distribute, collect, dot product) and class `SparseMatrix`, that contains the code for parallel matrix operations.

The distribution of matrix and vectors is determined by the global array `parallel.idomains^2`, which contains \( d(i) \) for each \( i \). This is initialized and distributed in the procedure `initialize()`.

Here we also calculate the mapping from local to global indices and store it to `local2global` array.

The matrix \( A \) is in triple format, where row and column indices are stored to the arrays `irows` and `icols`, and matrix values to the array `vals`. Before parallel matrix-vector multiplication is possible the following steps need to be done

1. distribute sparse matrix \( A \) and save it locally on each process into sparse matrices \( A_{local} \) and \( A_{ghost} \)

\[^2\text{In FDM domain } \Omega \text{ is split into subdomains } \Omega_i, \text{ so every process gets its own domain to work on which in our case just means a process has its part of } y \text{ to calculate}\]
• see `parallel.SparseMatrix._distribute()` (distribute data)

2. determine which $x$ values need to be received from other processes and notify them about it
   (prepare data for communication)
• see `parallel.SparseMatrix._prepare_communication()`

3. reindex $A_{\text{local}}$ and $A_{\text{ghost}}$ matrices, so that they can be multiplied with $x_{\text{local}}$ and $x_{\text{ghost}}$
   (reindexing)
• see `parallel.SparseMatrix._reindex_locally()`

1.2.1 Distribute data
The data of matrix $A$ is distributed using `parallel.idomains` array, the $A_{\text{local}}$ and $A_{\text{ghost}}$
matrices are stored to `_local_mat` and `_ghost_mat` fields. It is also not yet possible to use
these matrices, because the indices in the arrays `irows` and `icols` refer to the rows and columns
of matrix $A$.

Example  The `idomains` is initialized to $[0 \ 0 \ 1 \ 1]$.
   Let see local and ghost sparse matrices for process 0 and 1 ($A_{\text{local}}^{(\text{process}=0)}, A_{\text{ghost}}^{(\text{process}=0)}, A_{\text{local}}^{(\text{process}=1)},
   A_{\text{ghost}}^{(\text{process}=1)})$ after distribution. Notice, that row and column indices are still global, so we can’t
yet multiply them with locally stored vectors $x_{\text{local}}, x_{\text{ghost}}$.

```
local_mat, process=0 4 x 4 nnz=4
  0 0 1.000000000000000
  0 1 2.000000000000000
  1 0 3.000000000000000
  1 1 2.000000000000000
ghost_mat, process=0 4 x 4 nnz=1
  1 3 2.000000000000000
local_mat, process=1 4 x 4 nnz=3
  2 2 4.000000000000000
  2 3 -0.500000000000000
  3 3 2.000000000000000
ghost_mat, process=1 4 x 4 nnz=2
  2 1 -2.000000000000000
  3 0 1.000000000000000
```

1.2.2 Prepare data for communication
On each process calculate $x$ indices we need from other processes and store them in `_from_neighs`
field. Exchange the indices by using mpi4py `alltoall()` and store the indices we need to send to other
processes in `_to_neighs` field. After this operation `_from_neighs[i]` contains indices of global
vector $x$, that current process needs from process $i$. 

4
After exchanging the information about ghost data, that needs to be exchanged the 
_from_neighs and _to_neighs lists of arrays are the following.

\[
\text{from}_\text{neighs}, \text{process}=0 \quad \text{[array([], dtype=int64), array([3])]} \\
\text{from}_\text{neighs}, \text{process}=1 \quad \text{[array([0, 1]), array([], dtype=int64)]} \\
\text{to}_\text{neighs}, \text{process}=0 \quad \text{[array([], dtype=int64), array([0, 1])]} \\
\text{to}_\text{neighs}, \text{process}=1 \quad \text{[array([3]), array([], dtype=int64)]}
\]

The data tells that process 0 needs $x_3$ from process 1 and process 1 needs $x_0, x_1$ from process 0. Again, all indices are currently global.

From the _from_neighs list we can see, which $x$ values will be stored in $x_{\text{ghost}}$ of each process, particularly $x_{\text{ghost}}^{(\text{process}=0)} = (x_3)$ and $x_{\text{ghost}}^{(\text{process}=1)} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$. If there are more number of processes, then index arrays from _from_neighs may be stacked up vertically to form $x_{\text{ghost}}^{(\text{process}=i)}$ on each process.

### 1.2.3 Reindexing

The last step is to reindex irows, icols of both matrices and _from_neighs and _to_neighs using global2local array, which maps global indices to local indices of $x_{\text{local}}$ and ghost indices of $x_{\text{ghost}}$. Do not forget to change $m$ and $n$ values of the matrices. This will allow to use sparse matrix operations from the sparse Python module directly.

**Example** After reindexing the matrices $A_{\text{local}}^{(\text{process}=0)}, A_{\text{ghost}}^{(\text{process}=0)}, A_{\text{local}}^{(\text{process}=1)}, A_{\text{ghost}}^{(\text{process}=1)}$ can be used in $A_{\text{local}}x_{\text{local}}, A_{\text{ghost}}x_{\text{ghost}}$ operations directly. Compare the following row and column indices with the corresponding indices before reindexing (see section 1.2.1).

\[
\text{local}_\text{mat}, \text{process}=0 \quad 2 \times 2 \quad \text{nnz}=4 \\
0 \ 0 \ 1.000000000000000 \\
0 \ 1 \ 2.000000000000000 \\
1 \ 0 \ 3.000000000000000 \\
1 \ 1 \ 2.000000000000000 \\
\text{ghost}_\text{mat}, \text{process}=0 \quad 2 \times 1 \quad \text{nnz}=1 \\
1 \ 0 \ 2.000000000000000 \\
\text{local}_\text{mat}, \text{process}=1 \quad 2 \times 2 \quad \text{nnz}=3 \\
0 \ 0 \ 4.000000000000000 \\
0 \ 1 \ -0.500000000000000 \\
1 \ 1 \ 2.000000000000000 \\
\text{ghost}_\text{mat}, \text{process}=1 \quad 2 \times 2 \quad \text{nnz}=2 \\
0 \ 1 \ -2.000000000000000 \\
1 \ 0 \ 1.000000000000000 \\
\]

### 1.2.4 Parallel sparse matrix-vector multiplication

The operation is straightforward – use 3 steps described in section 1.1.1.
1.3 Task description

1. Implement reindexing
2. Implement parallel matrix-vector multiplication