J. H. Walther
S. Martin
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2022

Set 6 - Sparse Linear Algebra with MPI

Issued: December 07, 2022
Hand in (optional): December 20, 2022 23:59

Question 1: CSR format (10 points)

Sparse matrices have a limited number of nonzero elements. Specialized formats are used for efficient storage and operations on them. The CSR format (compressed sparse row) represents a matrix A_{ij} by three one-dimensional arrays:

- $A[k]$, nonzero coefficients in the row-major order, $0 \leq k < N_z$, where N_z is the number of non-zero elements of the matrix;
- $K[i]$, extents of rows, row i consists of $A[K[i] : K[i+1]]$;
- $J[k]$, column indices.

Write down a representation of the following matrix in the CSR format

$$A = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -2 & 4 & -2 \\ -1 & 0 & -1 & 2 \end{bmatrix}.$$

From that representation, compute the product Au with column $u = [0, 1, 2, 3]^T$.

The CSR representation:

```
A:  2  -1  -1  -1   2  -1  -2  4  -2  -1  -1   2
K:  0   3   6   9  12
J:  0   1   3   0   1   2   1  2   3   0   2   3
```

The matrix-vector product $b = Au = [-4 \;\; 0 \;\; 0 \;\; 4]^T$.

- **1 point** for correct final result of matrix-vector product
- **3 points** "A" in correct CSR format (-1 point per wrong element, for a minimum of 0)
- **3 points** "K" in correct CSR format (-1 point per wrong element, for a minimum of 0)
- **3 points** "J" in correct CSR format (-1 point per wrong element, for a minimum of 0)

Question 2: Matrix-vector product with MPI (90 points)

Matrices and vectors can be distributed to multiple processors. Consider the matrix-vector product

$$b = Au$$

with a square matrix $A \in \mathbf{R}^{n \times n}$ and vectors (columns) $u, b \in \mathbf{R}^n$. Each of p processors stores n/p rows of the matrix, the corresponding chunks of vector b and the same chunks of vector u. For dense matrices (many non-zero elements), the communication pattern is all-to-all. However, for a banded matrix (e.g. tridiagonal) communication is required only between neighbouring chunks. Such sparse matrices (most elements are zero) result from the discretization of Partial Differential Equations (PDEs).

One example is the two dimensional diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

which can be discretized as

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{\Delta t}{h}(u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n - 4u_{i,j}^n)$$

for $i = 0, \ldots, N_i - 1$, $j = 0, \ldots, N_j$. This can be rewritten as a matrix-vector product $u^{n+1} = u^n + Au^n$, by defining

$$u^n = [u_{0,0}^n, u_{0,1}^n, \ldots, u_{0,N_j-1}^n, u_{1,0}^n, u_{1,1}^n, \ldots, u_{1,N_j-1}^n, \ldots\ldots u_{N_i-1,0}^n, u_{N_i-1,1}^n, \ldots, u_{N_i-1,N_j-1}^n]^T.$$

a) Derive matrix A so that the matrix-vector product $u^{n+1} = u^n + Au^n$ is equivalent to the discrete diffusion equation above. Matrix A is known as the Poisson matrix. Assume periodic boundary conditions by *defining*

$$\begin{aligned}
u_{N_i,j} &:= u_{0,j} && , j = 0, \ldots, N_j - 1 \\
u_{-1,j} &:= u_{N_i-1,j} && , j = 0, \ldots, N_j - 1 \\
u_{i,N_j} &:= u_{i,0} && , i = 0, \ldots, N_i - 1 \\
u_{i,-1} &:= u_{i,N_j-1} && , i = 0, \ldots, N_i - 1
\end{aligned}$$

The 2D Poisson matrix with periodic boundary conditions that corresponds to vector u^n is

$$A = \frac{\Delta t}{h} \begin{bmatrix}
-D & I & 0 & 0 & 0 & \ldots & 0 & I \\
I & -D & I & 0 & 0 & \ldots & 0 & 0 \\
0 & I & -D & I & 0 & \ldots & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & 0 & I & -D & I & 0 \\
0 & 0 & \ldots & 0 & 0 & I & -D & I \\
I & 0 & \ldots & 0 & 0 & 0 & I & -D
\end{bmatrix} \tag{1}$$

where $I \in \mathbb{R}^{N_j \times N_j}$ is the identity matrix and $D \in \mathbb{R}^{N_j \times N_j}$ is given from

$$D = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & \dots & 0 & -1 \\ -1 & 4 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & \dots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & \dots & 0 & 0 & -1 & 4 & -1 \\ -1 & 0 & \dots & 0 & 0 & 0 & -1 & 4 \end{bmatrix} \tag{2}$$

The elements marked with red are the ones that represent the periodic boundary conditions; different boundary conditions would need different values instead.

- **6 points** for correct matrix dimensions (dimensions of A,D and I)
- **6 points** for correct boundary conditions (-3 pts if boundary conditions were accounted for, but the corresponding elements in the matrix were wrong)
- **4 points** for correct values $(4, -1)$ in the matrix
- **4 points** for correct location of those values in the matrix

b) You are given a skeleton code that solves the diffusion equation by treating it as a series of matrix-vector products. Parallelize the sparse matrix-vector product with MPI. See the skeleton code for more details.

Hint: The algorithm should involve the following stages:

- traverse rows of the matrix, multiply the elements stored locally and collect global indices of columns that require communication; use `GlbToLoc()` to convert the global indices;
- use `GlbToRank()` to find the ranks of processors storing the required columns;
- use `MPI_Allreduce()` to compute the number of messages that every processor needs to receive;
- send and receive the indices of columns from other processors;
- send and receive the corresponding elements of vector u;
- add the received data to the product.

File `mul.h` in the solution code contains a reference implementation. For this question there are 60 points available.

- **20 points** if your solution gives the same result as the serial implementation for any number p of MPI processes (assuming the grid size if divisible by p).
- **10 points** if each MPI process only stored $N_i N_j / p$ elements of the solution vector. No points if each process owned the whole vector.
- **10 points** for taking care of multiplication of local elements seperately from multiplication of elements owned by other processes.
- **10 points** for sending/receiving indices of elements required for the multiplication.
- **10 points** for sending/receiving values of elements required for the multiplication.

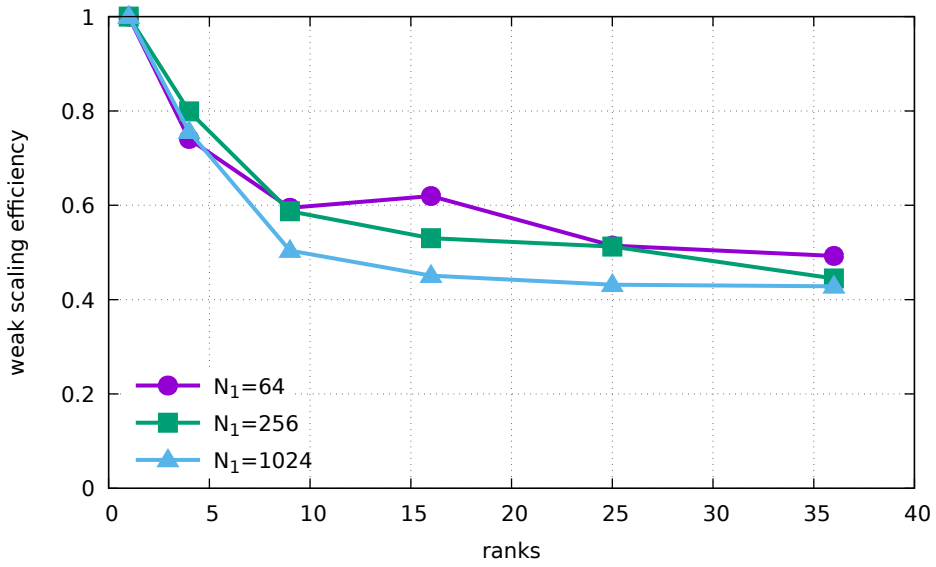

Figure 1: Weak scaling on Euler. N_1 is the grid size for one processor.

c) Report the weak scaling of your implementation on Euler.

For the weak scaling the number of components (i.e. number of grid cells) per processor should be kept constant. Fig. 1 shows the weak scaling efficiency up to 36 ranks on Euler.

- **5 points** if at least three different values of processors were used
- **5 points**if the number of grid points per processor was kept constant (no points if the timesteps per processor were kept constant)