Distributed Arnoldi Method for Eigenvalues using PETSc & Lapack

Antonio De Lucreziis

Abstract. In this project we implement the Arnoldi Method for eigenvalues using Lapack, PETSc, OpenMPI libraries for distributed matrix and vector operations and run some numerical simulations on a small cluster of ~30 nodes of Rock 4C+ (`aarch64`) boards we built for the course of High-Performance Mathematics. The code can be found at github.com/aziis98/arnoldi-distribuito.

Contents

1. Arnoldi Method for Eigenvalues

1.1. Theory of Krylov subspaces

Definition 1. The Krylov subspace of order ℓ associated with A and b is the subspace

$$\mathcal{K}_\ell(A, b) \coloneqq \operatorname{span}\big(b, Ab, ..., A^{\ell-1}b\big)$$

Definition 2. If x_ℓ is an approximation of the solution to $Ax = b$ then the residual is defined as

$$r_\ell \coloneqq b - Ax_\ell$$

There are various criteria that can be used to identify the "optimal" solution

- **FOM** – Have the residual be orthogonal to $\mathcal{K}_{\ell(A,b)}$
- **GMRES** – Have the residual minimized such that x_ℓ is

$$x_\ell \coloneqq \operatorname{argmin}_{x \in \mathcal{K}_{\ell(A,b)}} \|b - Ax\|_2$$

We are now interested in studying the **FOM** case and its behavior when $\ell \ll n$.

1.2. Arnoldi Iteration

We now want to construct a nice basis for $\mathcal{K}_{\ell(A,b)}$, as the one directly given by $A^j b$ has very poor computational properties. We need a way of constructing an orthogonal basis of the Krylov subspace without using the QR algorithm.

Let's define the Arnoldi iteration that works as follows:

- First let's define $v_1 \coloneqq b/\|b\|_2$
- For $j = 1, ..., \ell$
  - Let's define $w_{j+1} \coloneqq Av_j$
  - The to obtain the next vector let's orthogonalize it to the current basis $v_1, ..., v_j$ and let

$$\pi_{j(w_{j+1})} \coloneqq \sum_{i \leq j} (v_i^* w_{j+1}) v_i$$

$$v_{j+1} \coloneqq \frac{w_{j+1} - \pi_{j(w_{j+1})}}{\left\| w_{j+1} - \pi_{j(w_{j+1})} \right\|_2}$$

Let's also define the following **upper Hessenberg matrix** composed of all the projection coefficients

$$h_{i,j} \coloneqq v_i^* w_{j+1} = v_i^* A v_j$$

$$H_\ell \coloneqq \begin{pmatrix} h_{1,1} & \dots & \dots & h_{1,\ell} \\ h_{2,1} & h_{2,2} & & \vdots \\ & \ddots & \ddots & \vdots \\ & & h_{\ell,\ell-1} & h_{\ell,\ell} \end{pmatrix}$$

The final algorithm for the **Arnoldi iteration** is

function ARNOLDI-ITERATION(A, b, ℓ)
    $q_1 \leftarrow b/\|b\|_2$
    for $k = 2, .., \ell$ **do**
        $q_k \leftarrow Aq_{k-1}$
        for $j = 1, ..., k-1$ **do**
            $h_{j,k-1} \leftarrow q_j^* q_k$
            $q_k \leftarrow q_k - h_{j,k-1} q_j$
        $h_{k,k-1} \leftarrow \|q_k\|_2$
        $q_k \leftarrow q_k / h_{k,k-1}$

1.3. Arnoldi iteration for eigenvalues and eigenvectors

To use the Arnoldi iteration to compute the eigenvalues of A we can use the following procedure:

- First we use the Arnoldi iteration to compute the upper Hessenberg matrix H_ℓ for the Krylov subspace $\mathcal{K}_{\ell(A,b)}$.
- We then compute the eigenvalues and eigenvectors of this Hessenberg matrix by computing a QR decomposition of the matrix H_ℓ.

The following theorem states the relationship between the eigenvalues of H_ℓ and A

Theorem 1. *Let H_ℓ be the upper Hessenberg matrix from the Arnoldi process for A (without breakdown), starting from b. Then, the characteristic polynomial $\hat{p}(\lambda) = \det(\lambda I - H_\ell)$ satisfies*

$$\hat{p} = \underset{\substack{p \in \mathcal{P}_\ell \\ p_\ell = 1}}{\operatorname{argmin}} \|p(A)b\|_2$$

So the characteristic polynomial of H_ℓ is a good approximation of the characteristic polynomial of A.

To benchmark the performance of our implementation of the distributed Arnoldi method we will use the test problem of finding the eigenvalues of the 3D Laplacian that has a nice closed form solution.

1.4. **3D Laplacian**

The matrix we will consider comes up from the Laplace equation. Let $u : \mathbb{R}^n \to \mathbb{R}$, for example we can consider the zero Dirichlet boundary condition on the unit cube $[0,1]^n$ and get the following problem

$$\begin{cases} \Delta u(x) = 0 & x \in [0,1]^3 \\ u(x) = u_0(x) & x \in \partial[0,1]^3 \end{cases}$$

We now discretize this problem using the finite difference method. We will use a uniform grid with n_i points in the i-th direction (and also let $h_i := 1/(n_i + 1)$). In dimension $n = 1$ the problem is just the second derivative of u with respect to x and the finite difference approximation is

$$D_{xx}u(x) = \frac{u(x + h_x) - 2u(x) + u(x - h_x)}{h_x^2}$$

where $h_x := 1/(n_1 + 1)$ is the distance between two consecutive points. Notice that here the stencil has shape $[1 \ -2 \ 1]$. If we consider the matrix $\mathbf{D}_{xx}$ associated with the finite difference operator, we can write the problem as $\mathbf{D}_{xx}\overline{u} = 0$ where $\overline{u}$ is the vector of values of u at the grid points, and $\mathbf{D}_{xx}$ is the following tridiagonal matrix

$$\mathbf{D}_{xx} = \frac{1}{h_1^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}$$

It is well known that a tridiagonal matrix with constant diagonals of a, b, c has eigenvalues

$$\lambda_k = a - 2\sqrt{bc}\cos\left(\frac{\pi k}{n+1}\right)$$

for $k = 1, ..., n$. In this case we have $a = -2$, $b = 1$, $c = 1$ so the eigenvalues are given by the following expression (and have the following bounds):

$$\begin{aligned}\lambda_k &= -2 - 2\cos\left(\frac{\pi k}{n+1}\right) \\ &= -2\left(1 + \cos\left(\pi\frac{k}{n+1}\right), \in [-1,1]\right) \\ &= \underbrace{-4\cos^2\left(\frac{\pi}{2}\cdot\frac{k}{n+1}\right)}_{\in[0,1]} \Longrightarrow -4 < \lambda_k < 0\end{aligned}$$

For dimension $n = 2$ we have the following stencils:

$$\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix} = \begin{bmatrix}1 & -2 & 1\end{bmatrix} + \begin{bmatrix}1\\-2\\1\end{bmatrix}$$

$$\Updownarrow$$

$$\big(D_{xx} + D_{yy}\big)u(x,y) = \frac{u(x-h_1,y) + u(x+h_1,y) + u\big(x,y-h_y\big) + u\big(x,y+h_y\big) - 4u(x,y)}{h_1^2}$$

In this case the matrix associated with the finite difference operator can be written compactly as the sum of two Kronecker products:

$$D_{xx} + D_{yy} = \mathbf{I} \otimes \mathbf{D}_{yy} + \mathbf{D}_{xx} \otimes \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix.

This can also be generalized to the case of $n = 3$ and higher dimensions. The matrix for the discretized 3D Laplacian can be written as

$$L = \mathbf{D_{xx}} \otimes \mathbf{I} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{D_{yy}} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{I} \otimes \mathbf{D_{zz}}$$

We are now interested in finding the eigenvalues of this matrix for large N, we will drop the factor $1/h_i^2$ from the matrix and directly compute the eigenvalues of the matrix L.

To compute the eigenvalues and eigenvectors of this matrix we can use the following facts about the Kronecker products.

Proposition 2. *The Kronecker product has the following properties*

- *The Kronecker product distributes over the matrix product:*
$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$$
- *Conjugation commutes with the Kronecker product, i.e.*
$$(A \otimes B)^* = A^* \otimes B^*$$
- *Let $A = A_1 \otimes A_2 \otimes ... \otimes A_n$ be the Kronecker product of n matrices. Then, the eigenvalues of A are given by the products of the eigenvalues of the A_i.*

Proposition 3. *Given two diagonalizable matrices A and B with eigenvalues λ_i and μ_j, respectively the expression*

$$A \otimes I + I \otimes B$$

has eigenvalues $\lambda_i + \mu_j$ and a matrix of eigenvectors is $V \otimes W$ where V and W are the matrices of eigenvectors of A and B, respectively

$$A = V D_A V^* \qquad B = W D_B W^*$$

$$D_A = \begin{bmatrix} \ddots & & \\ & \lambda_i & \\ & & \ddots \end{bmatrix} \qquad D_B = \begin{bmatrix} \ddots & & \\ & \mu_j & \\ & & \ddots \end{bmatrix}$$

Proof. Let's consider $D_A = V^* D_A V$ and $D_B = W^* D_B W$. Then, by conjugation with $V \otimes W$ we have

$$\begin{aligned}
&(V \otimes W)^*(A \otimes I + I \otimes B)(V \otimes W) \\
&= (V^* \otimes W^*)(A \otimes I + I \otimes B)(V \otimes W) \\
&= (V^* \otimes W^*)(A \otimes I)(V \otimes W) + (V^* \otimes W^*)(I \otimes B)(V \otimes W) \\
&= (V^* A V \otimes I) + (I \otimes W^* B W) \\
&= D_A \otimes I + I \otimes D_B
\end{aligned}$$

To conclude let's analyze the block structure of this last term

$$D_A \otimes I + I \otimes D_B = \begin{bmatrix} \ddots & & \\ & \lambda_i \boxed{I} & \\ & & \ddots \end{bmatrix} + \begin{bmatrix} \ddots & & \\ & \boxed{D_B} & \\ & & \ddots \end{bmatrix} = \begin{bmatrix} \ddots & & \\ & \boxed{\lambda_i I + D_B} & \\ & & \ddots \end{bmatrix}$$

We can thus produce all possible combinations of sums of λ_i and μ_j and so the final eigenvalues are given by

$$= \begin{bmatrix} \ddots & & \\ & \lambda_i + \mu_j & \\ & & \ddots \end{bmatrix}$$

□

This fact can also be generalized to the case of n dimensions. Recalling that the eigenvalues of the 1D Laplacian are given by

$$\lambda_i = -4 \cos^2\left(\frac{\pi i}{2(n+1)}\right)$$

we can write the eigenvalues of the 3D Laplacian as

$$\lambda_{i_1,i_2,i_3} = -4\left(\cos^2\left(\frac{\pi i_1}{2(n_1+1)}\right) + \cos^2\left(\frac{\pi i_2}{2(n_2+1)}\right) + \cos^2\left(\frac{\pi i_3}{2(n_3+1)}\right)\right)$$

for $(i_1, i_2, i_3) \in [1, ..., n_1] \times [1, ..., n_2] \times [1, ..., n_3]$. This is approximately in the range $[-12, 0]$.

2. PETSc Implementation

Let's look again at the algorithm for the Arnoldi iteration. There are various operations that can be done in parallel.

1: **function** ARNOLDI-ITERATION(A, b, ℓ)
2: $q_1 \leftarrow b/\|b\|_2$
3: **for** $k = 2, .., \ell$ **do**
4: $q_k \leftarrow Aq_{k-1}$
5: **for** $j = 1, ..., k-1$ **do**
6: $h_{j,k-1} \leftarrow q_j^* q_k$
7: $q_k \leftarrow q_k - h_{j,k-1} q_j$
8: $h_{k,k-1} \leftarrow \|q_k\|_2$
9: $q_k \leftarrow q_k / h_{k,k-1}$

If we store the matrix A and the vector b in a distributed way, we can compute the following operations in parallel:

- The normalization of b using the function `VecNormalize`.
- The matrix-vector product Aq_k using `MatMult`.
- The inner product $q_j^* q_k$ using `VecDot`.
- The operation $q_k - h_{j,k-1} q_j$ can be done in parallel as well. More precisely, this can be done as a *fused multiply-add* operation using the PETSc `VecAXPY` function.
- The norm of q_k can be computed using `VecNorm`.

The final algorithm written in C is the following: we will omit all calls to `PetscCall(...)` and other PETSc related infrastructure for the sake of brevity.

```c
PetscErrorCode ArnoldiIteration(Mat A, Vec b, PetscInt n, PetscInt m, Vec *Q, double *h) {
    Vec q;
    VecDuplicate(b, &q);
    VecCopy(b, q);
    VecNormalize(q, NULL);
    Q[0] = q;
    for (PetscInt k = 1; k < n + 1; k++) {
        Vec v;
        VecDuplicate(b, &v);
        MatMult(A, Q[k - 1], v);
        PetscScalar h_ij;
        for (PetscInt j = 0; j < k; j++) {
            VecDot(Q[j], v, &h_ij);
            h[j * (n + 1) + k - 1] = h_ij;
            VecAXPY(v, -h_ij, Q[j]);
        }
        VecNorm(v, NORM_2, &h_ij);
        h[j * (n + 1) + k - 1] = h_ij;
        if (h_ij > eps) {
            VecNormalize(v, NULL);
            Q[k] = v;
        } else {
            break; // Early breakdown
        }
    }
}
```

The matrix A is stored in a distributed way using the PETSc `Mat` type. The vector b is also stored in a distributed way using the PETSc `Vec` type. In the first experiments the matrix A is a 3D Laplacian matrix (symmetric case) and the vector b is the vector of ones.

3. Numerical Experiments

For the numerical experiments we will use a Krylov subspace of dimension $\ell = 25$. We will use the following test problems

- **Strong scaling** – Fixed size problem with varying number of processes. We expect this to start slow and get faster as we increase the number of processes up to a certain point where the overhead of communication will start to deteriorate the computation time.
- **Weak scaling** – Varying number of processes and varying size of the problem (constant amount of work per node). We expect this to start fast and get slower as we increase the size of the problem as the cost of communication will deteriorate the computation time.
- **SuiteSparse Experiment** – Varying number of processes on a matrix from the SuiteSparse matrix collection. We chose the matrix `atmosmodd` which has a symmetric pattern of non-zeros but is itself non-symmetric to see how the Hessenberg matrix is affected by the non-symmetry (we get a proper, non tridiagonal hessenberg matrix as with the symmetric Laplacian case).

3.1. Experiment: 3D Laplacian

All the sparse laplacian matrices are generated using the following Julia code, which generates the 3D Laplacian discretization for a grid of side N. The matrix is generated sparse using the `spdiagm` function from `SparseArrays`, `kron` is the Kronecker product function from `LinearAlgebra` and `matwrite` is a function from the `MAT` package to write the matrix to a file in the HDF5 format as needed by PETSc.

```
using LinearAlgebra
using SparseArrays
using MAT

N = parse(Int, ARGS[1])

ex = fill(1, N - 1)
ey = fill(1, N - 1)
ez = fill(1, N - 1)

Dxx = spdiagm(-1 => ex, 0 => -2 * [ex; 1], +1 => ex)
Dyy = spdiagm(-1 => ey, 0 => -2 * [ey; 1], +1 => ey)
Dzz = spdiagm(-1 => ez, 0 => -2 * [ez; 1], +1 => ez)

Ix = spdiagm(0 => [ex; 1])
Iy = spdiagm(0 => [ey; 1])
Iz = spdiagm(0 => [ez; 1])

L = kron(Dxx, Iy, Iz) + kron(Ix, Dyy, Iz) + kron(Ix, Iy, Dzz)

matwrite("laplacian_$N.mat", Dict("A" => L))
```

We dropped the factor $1/h_i^2$ from the matrix and computed its eigenvalues for a grid of size $N \times N \times N$. From the previous discussion we can see that the eigenvalues of the 3D Laplacian are given by the following expression for $i_1, i_2, i_3 = 1, ..., N$:

$$\lambda_{i_1,i_2,i_3} = -4\left(\cos^2\left(\frac{\pi i_1}{2(n_1+1)}\right) + \cos^2\left(\frac{\pi i_2}{2(n_2+1)}\right) + \cos^2\left(\frac{\pi i_3}{2(n_3+1)}\right)\right)$$

The factor $\cos^2(...)$ is bounded in $[0, 1]$ so the eigenvalues are bounded in $[-12, 0]$. First we can check the correctness of the implementation by checking the eigenvalues and eigenvectors returned by our implementation. In Figure 1 we can see the resulting Hessenberg matrix while in Figure 2 we have the Schur form returned by `dhesqr`.

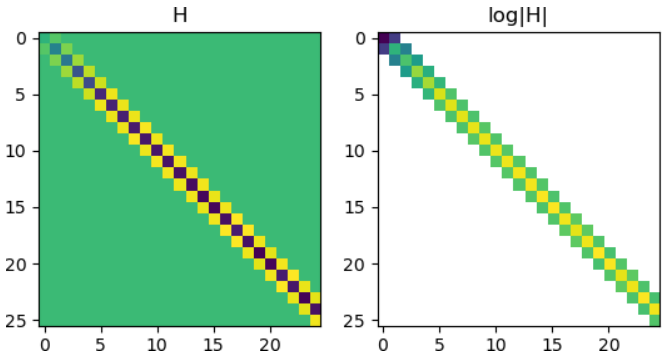

FIGURE 1. Hessenberg matrix for the 3D Laplacian with $N = 20$ and $\ell = 25$.

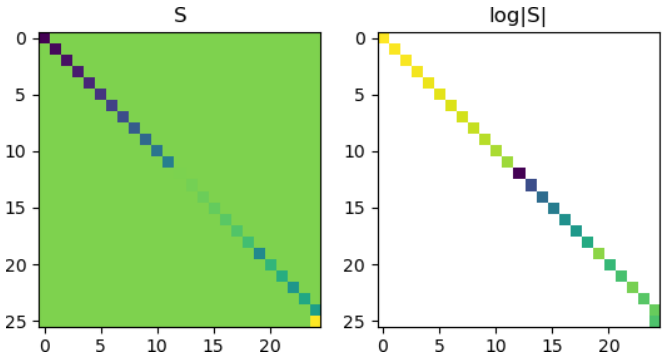

FIGURE 2. Schur matrix for the 3D Laplacian with $N = 20$ and $\ell = 25$ after calling `dhesqr`

We get the following eigenvalues:

$$\begin{aligned}\Lambda = \{&-11.73, -11.43, -11.07, -10.64, -10.13,\\ &-9.55, \ -8.91, \ -8.21, \ -7.47, \ -6.82,\\ &-6.16, \ -5.49, \ -4.81, \ -4.11, \ -3.59,\\ &-3.09, \ -2.64, \ -2.16, \ -1.61, \ -1.12,\\ &-0.91, \ \ -0.6, \ -0.43, \ -0.24, \ -0.07 \ \}\end{aligned}$$

3.1.1. **Strong Scaling**

First we tried to run the program starting from a problem size of $N = 20$ and increasing the problem size by doubling each time. As we can see in Figure 3 the performance is initially poor (for $N \in \{20, 40\}$) but starts to improve as we increase the size of the problem as initially the problem is too small and the communication overhead is too high.

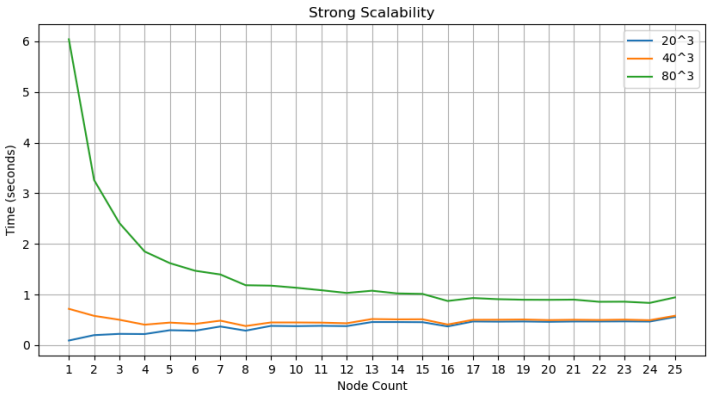

FIGURE 3. Times for varying node count with $N \in \{20, 40, 80\}$, $\ell = 25$.

We get our final result in Figure 4 for problem size up to $N = 160$ and we can see that the performance is quite good. The log-log plot in Figure 5 better shows the performance in all the range of problem sizes.

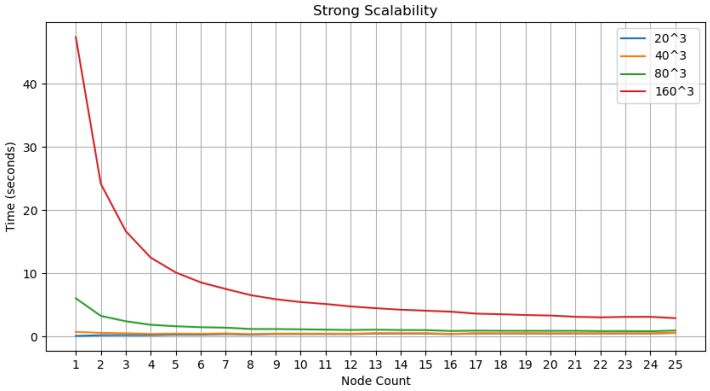

FIGURE 4. Times for varying node count with $N \in \{20, 40, 80, 160\}$ and $\ell = 25$.

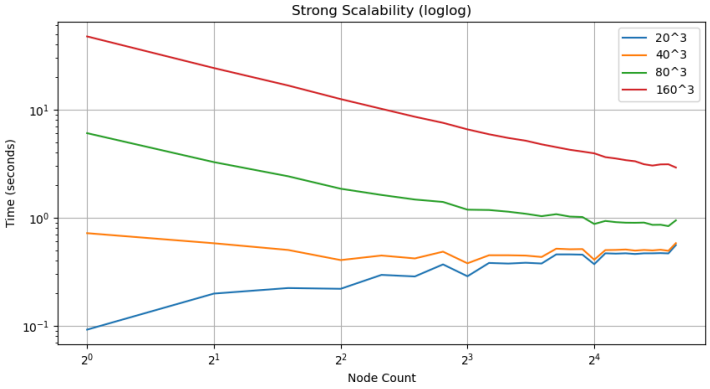

FIGURE 5. Times log-log for varying node count with $N \in \{20, 40, 80, 160\}$ and $\ell = 25$.

From these results we can also see how the performance stabilizes as we increase the node count and that at a certain point adding more nodes does not improve as much the performance.

3.1.2. **Weak Scaling**

For the weak scaling we generated the matrices for $N = \left\lceil \sqrt[3]{i \times 20000} \right\rceil$ for $i \in \{1, ..., 25\}$ to get a *constant amount of work per node*. We can see in Figure 6 that the performance degrades as we increase the number of nodes. This is expected as the problem size increases and the communication overhead becomes more significant.

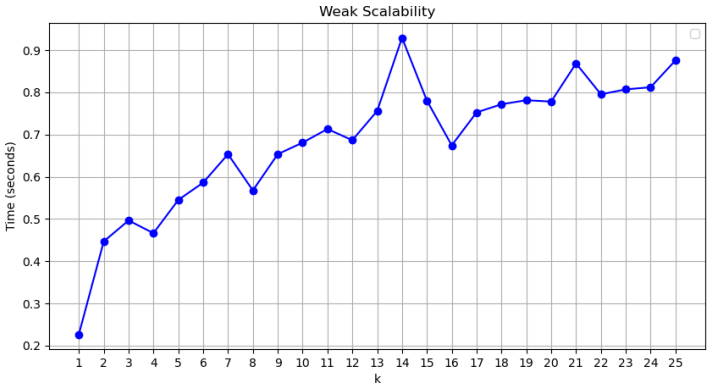

FIGURE 6. Times with $N = \left\lceil \sqrt[3]{i \times 20000} \right\rceil = \{28, ..., 80\}$ for $i \in \{1, ..., 25\}$ and $\ell = 25$.

3.2. **Experiment: `atmosmodd` from SuiteSparse**

We also evaluated the performance against a matrix from the SuiteSparse collection. We choose the `atmosmodd` matrix. It is a non-symmetric matrix with a symmetric pattern of non-zeros, here is some information about the matrix:

Name	atmosmodd
Group	Bourchtein
Matrix ID	2265
Num Rows	1'270'432
Num Cols	1'270'432
Non-zeros	8'814'880
Pattern Entries	8'814'880
Kind	Computational Fluid Dynamics Problem
Symmetric	No
Date	2009
Author	A. Bourchtein
Editor	T. Davis

Structural Rank	1'270'432
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	0
Pattern Symmetry	100%
Numeric Symmetry	66.9%
Cholesky Candidate	No
Positive Definite	No
Type	Real

In Figure 7 we can see that the resulting Hessenberg matrix looks tridiagonal but when looking at the log-log plot we actually see that the matrix is a proper Hessenberg matrix. The resulting Schur matrix is shown in Figure 8 and the computed eigenvalues are the following:

$$\Lambda = \{-187'078.67, -181'294.17, -173'634.23, -167'725.76, -160'474.49,$$
$$-152'024.13, -142'505.34, -132'037.86, -120'709.76, -108'473.08,$$
$$-93'768.3, \ -83'850.61, \ -75'095.74, \ -65'634.56, \ -56'074.01,$$
$$-46'728.52, \ -37'822.94, \ -29'538.87, \ -22'025.98, \ -15'371.79,$$
$$-9'907.26, \ -5'664.09, \ -2'596.04, \ -739.49, \ -60.07 \ \}$$

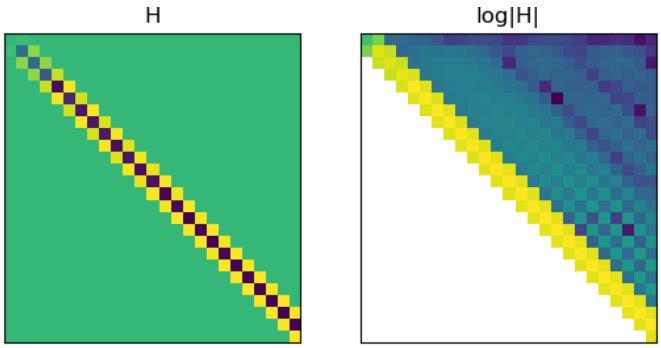

FIGURE 7. Hessenberg matrix for the SuiteSparse matrix `atmosmodd` with $\ell = 25$.

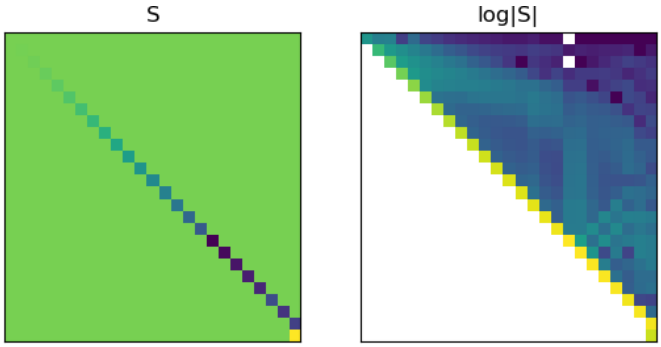

FIGURE 8. Schur matrix for the SuiteSparse matrix `atmosmodd` with $\ell = 25$ after calling `dhesqr`

Finally, we can see the same strong scaling performance for this non-symmetric matrix as for symmetric the 3D Laplacian.

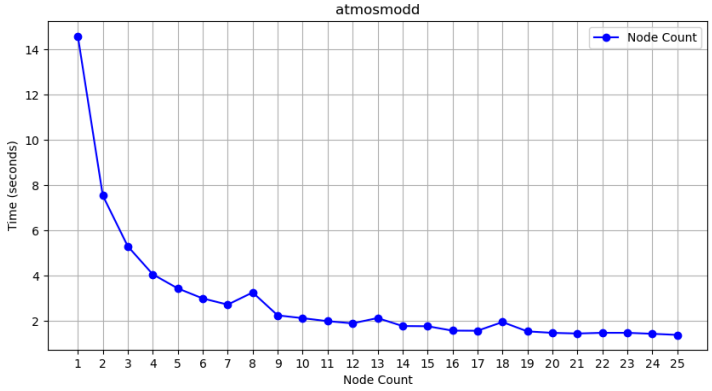

FIGURE 9. Times of the Arnoldi method for the SuiteSparse matrix `atmosmodd` with $\ell = 25$ with varying number of nodes.