Study and comparison of classical DDM Application to linear elasticity

By : Yizhi YANG, Yutai ZHAO
Link to Github

Contents

1 Introduction

In this project, our subject of study is a one-dimensional slender bar, with one end fixed and the other subjected to a known tensile force F_d. With a fixed cross-sectional area A and Young's modulus E, we will use different numerical methods to solve for the displacement and stress at various nodes and discuss the convergence of these methods. The length L is also known but it may vary during the subsequent scalability study. Note that in every algorithm implementation's description we tried to not repeat what is already written and coded, so only some formula is presented for reminding. The important points are the precisions and explanations of certain choices in the code.

2 Preliminary data setting work

2.1 Analytical solution

This section analytically solves the $1D$ elastic bar problem under tensile force using equilibrium conditions, boundary constraints, and Hooke's law. The solution shows that the stress remains constant throughout the bar, the strain is uniform, and the displacement increases linearly from the fixed end to the free end, reaching its maximum at the point of applied force.
We obtain finally the analytical solution of this problem:

$$\sigma_{xx}(x) = \frac{F_d}{A}$$
$$\epsilon_{xx}(x) = \frac{F_d}{EA}$$
$$u_x(x) = \frac{F_d}{EA}x$$

2.2 Carry out a finite element bar code

Finite element We discretize the bar with N elements, thus $n = N + 1$ nodes, with an identical distance $h = \frac{L}{N}$ between each pair of neighboring nodes. Using the principle of minimum potential energy, we express the displacement field in terms of shape functions and nodal displacements. The strain is obtained as the spatial derivative of displacement, and the stress follows from Hooke's law. By integrating the strain energy density over the element's volume, the element stiffness matrix is obtained as:

$$K^e = \frac{EA}{h} \cdot \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

where E is Young's modulus, A is the cross-sectional area, and h is the element length. The global stiffness matrix is then assembled by summing contributions from all elements, ensuring nodal continuity across shared nodes. The system is solved under boundary conditions, where the first node is fixed ($u_1 = 0$) and an external force (F_d) is applied at the last node. This results in a

tri-diagonal matrix system that governs the nodal displacements.

$$\frac{EA}{h} \cdot \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & \cdot & 0 & 0 & 0 & 0 \\ 0 & \cdot & \cdot & \cdot & 0 & 0 & 0 \\ 0 & 0 & \cdot & \cdot & \cdot & 0 & 0 \\ 0 & 0 & 0 & \cdot & \cdot & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}_{n \times n} \cdot \begin{bmatrix} 0 \\ u_2 \\ \cdot \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} f_1 \\ 0 \\ \cdot \\ 0 \\ F_d \end{bmatrix}$$

2.3 Direct solution

We implemented the following three direct solution methods in code, and these results will serve as a reference to verify the accuracy of the results obtained from the subsequent domain decomposition method.

2.3.1 Direct elimination of DOF

The system is reduced by directly removing the row and column corresponding to the constrained degree of freedom (DOF) where $u_1 = 0$. This results in a smaller system of equations, which can be solved iteratively.

2.3.2 Penalty method

A penalty term g is introduced into the system to enforce the constraint approximately. A large value of g ensures that the constrained DOF remains nearly fixed. The system is then solved iteratively, and the reaction force is computed as $f_1 = -g \times \frac{EA}{h} \times u_{1,approx}$.

2.3.3 Lagrangian multiplier method

An additional equation is introduced by adding a Lagrange multiplier λ to enforce the constraint exactly. This leads to an augmented system, which is then solved directly. The reaction force is obtained as $f_1 = -\lambda$.

2.4 Domain decomposition method

The L-length bar is decomposed into substructures S. A substructure of length H contains $N_s = \frac{N}{S}$ finite elements. A finite element of length h has 2 nodes. There are therefore $n_s = N_s + 1$ nodes pertaining to a substructure. The stiffness matrix for each substructure is quite similar to the global one, with the dimension changed to $n_s \times n_s$. Let's reorder the matrix for the bounding nodes (subscript $\square_b$) and inner nodes (subscript $\square_i$) and obtain:

$$k^{(s)} \cdot u^{(s)} = \begin{bmatrix} k_{ii}^{(s)} & k_{ib}^{(s)} \\ k_{bi}^{(s)} & k_{bb}^{(s)} \end{bmatrix} \cdot \begin{bmatrix} u_i^{(s)} \\ u_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} \end{bmatrix} = f^{(s)}$$

For this $1D$ problem, the dimension of k_{bb} is 1×1 for the first and the last substructure, and 2×2

for the rest (which corresponds to the number of rows and columns to permute). The local Primal Schur Complement matrix $S_p^{(s)}$ is therefore defined:

$$S_p^{(s)} = k_{bb}^{(s)} - k_{bi}^{(s)}[k_{ii}^{(s)}]^{-1}k_{ib}^{(s)}$$

For the first substructure, instead of using the whole matrix, we can eliminate the first row and column so that we use the matrix $k_{trim}^{(1)}$ to simplify the calculation of $S_p^{(1)}$.
The Dual Schur Complement $S_d^{(s)}$ is the generalized inverse of $S_p^{(s)}$:

$$S_d^{(s)} = {S_p^{(s)}}^+$$

The Primal RHS $b_p^{(s)}$:

$$b_p^{(s)} = f_b^{(s)} - k_{bi}^{(s)}[k_{ii}^{(s)}]^{-1}f_i^{(s)}$$

The rigid body modes $R_b^{(s)}$: $S_p^{(1)}R_b^{(1)} = 0$
However, the scalar $S_p^{(1)} \neq 0$, so $R_b^{(1)}$ has to be 0, but the kernel cannot be 0. There are consequently no rigid body modes for subdomain ($s = 1$).
For $s \in [2, S-1]$, the Primal right-hand side $b_p^{(s)}$:

$$b_p^{(s)} = f_b^{(s)} - k_{bi}^{(s)}[k_{ii}^{(s)}]^{-1}f_i^{(s)} = \vec{0}$$

Finding rigid body modes $R_b^{(s)}$ consists to solve: $S_p^{(s)}R_b^{(s)} = 0$
For $s = S$, we can therefore compute the Primal Schur Complement $S_p^{(s)}$ in using the trimmed matrix and obtain:

$$S_p^{(S)} = 0$$

Then the Dual Schur Complement, as the generalized inverse of $S_p^{(s)}$ is also 0, and the Primal right-hand side $b_p^{(S)} = F_d$. The rigid body modes: $S_p^{(S)}R_b^{(S)} = 0$
As $R_b^{(S)} \in \mathbb{R}^*$, we can choose $R_b^{(S)} = 1$.
The primal assembly operator $A^{(s)}$ is defined for each substructure, where the first and last subdomains have a single interface node, while the intermediate subdomains ($s \in [2, S-1]$) have two interface nodes. The concatenated primal assembly operator is then constructed as

$$\mathbb{A}^\diamond = [A^{(1)} \quad A^{(2\ldots S-1)} \quad A^{(S)}]_{(S-1)\times(2S-2)}$$

accounting for the total number of interface nodes $1 + 2(S-2) + 1 = 2S - 2$.
The primal complement matrix $S_p^\diamond$ is block diagonal, where each block corresponds to the Schur complement of a substructure. The primal right-hand side $b_p^\diamond$ is similarly defined, with a nonzero contribution only in the last entry (F_d).

On the other hand, the dual assembly operator $\underline{A}^{(s)}$ follows a similar structure but with sign differences, and the concatenated dual assembly operator is given by

$$\underline{\mathbb{A}}^{\diamond} = [\underline{A}^{(1)} \quad \underline{A}^{(2\ldots S-1)} \quad \underline{A}^{(S)}]_{(S-1)\times(2S-2)}$$

The concatenated primal complement matrix $S_p^{\diamond}$ is block diagonal, where each block corresponds to the Schur complement $Sp^{(s)}$ of a substructure, analogically for concatenated dual complement matrix $S_d^{\diamond}$. The primal right-hand side $b_p^{\diamond}$ is similarly defined, with a nonzero contribution only in the last node (F_d).
Additionally, the rigid body mode matrix $R_b^{\diamond}$ is structured such that the first subdomain does not have a rigid body mode, while the remaining subdomains contribute their respective rigid body modes. This matrix has $S-1$ columns, corresponding to subdomains with rigid body modes.

These presented elements will be used in the following sections, the construction details can be found in the code or in appendix where the whole construction process is detailed.

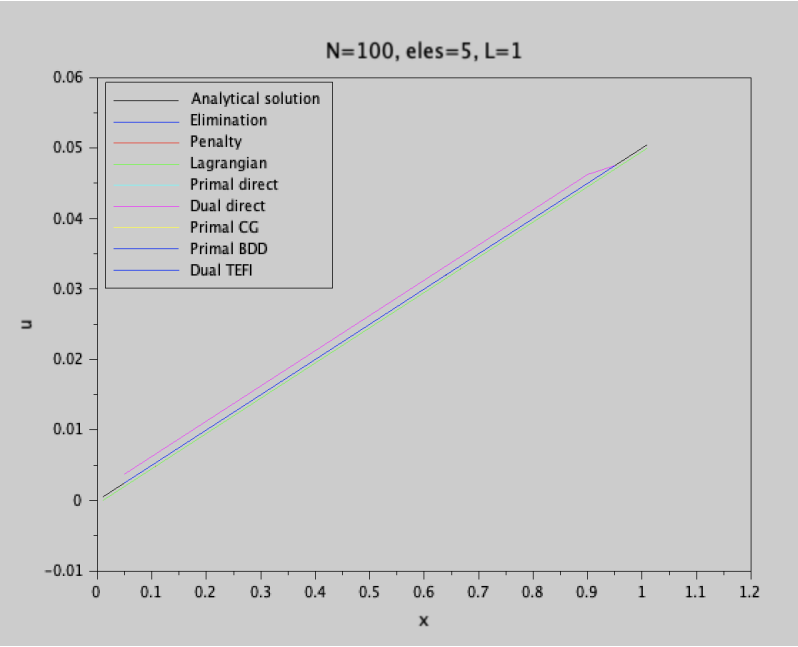

Figure 1: Displacement computed by different methods with total nodes = 100, subdomain elements =5 and bar length = 1

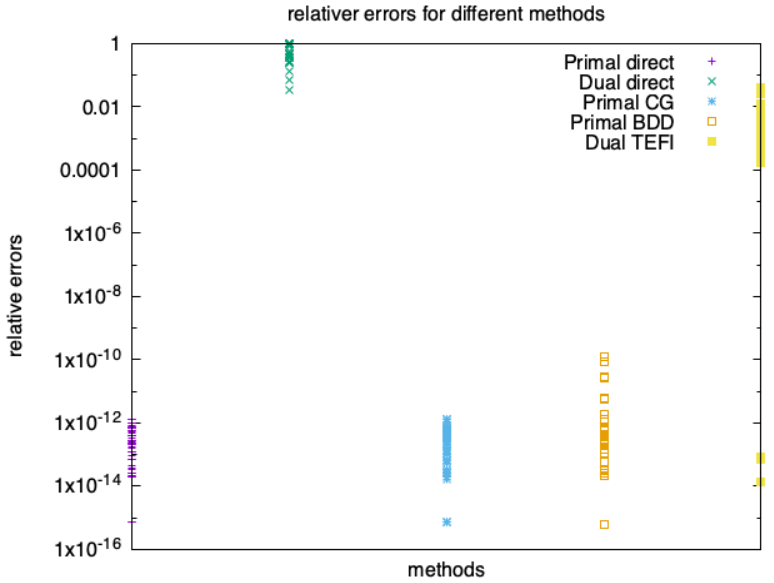

Figure 2: Relative errors for different methods

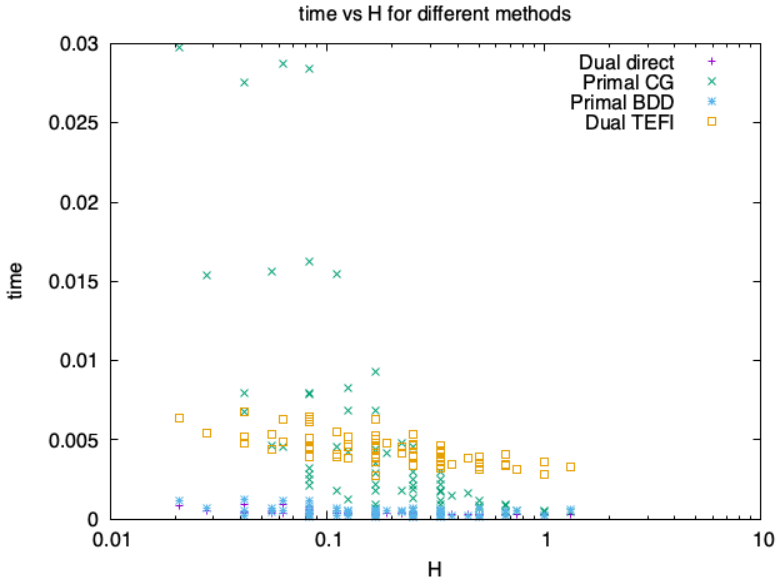

Figure 3: Parallel scalability : Computation Time with different H

3 Primal Schur approach

3.1 Direct method

3.1.1 Implementation

The `Primal_direct` function calculates the displacement at the interface by first constructing the concatenated Primal Schur Complement $S_p^\diamond$ and the transformation matrix $\mathbb{A}$. It then calculates the force vector $b_p^\diamond$ based on the applied forces at the boundary. The assembled Primal Schur matrix is built with $\mathbb{S}_p = \mathbb{A}S_p^\diamond\mathbb{A}^T$, and the right-hand side is transformed with $\mathbf{b}_p = \mathbb{A} * b_p^\diamond$. Finally, the displacement at the interface U_p is solved directly by $U_p = \mathbb{S}_p^{-1}\mathbf{b}_p$.

3.1.2 Correctness

We can visualize our results output, but a more rigorous way to verify the accuracy of our computation is to compute the relative error ($e = \frac{\|\hat{x}-x\|}{\|x\|}$), with x the reference vector and $\hat{x}$ the approached solution vector.

To have a more general vision, we also varied the length of the bar L, and the length of elements h for usual FE resolutions or the length of subdomains H for primal and dual resolutions. The variation of H is done by adjusting N the total number of nodes and eles the number of nodes in a subdomain. So in Fig.fig2 the points of the same color represent the correctness of different L,h or L,H of the same method.
As we have mentioned in the section 2.3, the direct solution is used as a reference to check the correctness of our approach. Finally, to make sure that we have the correct dimension of both these vectors, we only take the corresponding interfacial components of the direct solution (`u_ref`).
The same method of correctness check is applied for the sections 3.2.2 3.3.2 4.1.2 4.2.2 as well.

Come back to the correctness of primal direct resolution in fig2, we can see that it gives relatively good solutions.

3.1.3 Conditioning

The figure 1 show the conditioning of the assembled global stiffness matrix, K, of the non-substructure problem according to h.

For the conditioning of K, we can see that the curve of Direct elimination is reduced to 0 showing that its stiffness matrix is well conditioned. Langrage multiplier method also gives a well conditioned matrix when the element length is not too small. Finally Penality method gives a line at nearly 5e07 meaning that it gives an ill-conditioned K compared to the previous method. Knowing that a ill conditioned matrix, is very sensitive to any perturbations or rounding errors. It also suggests that the matrix is nearly singular (i.e. its determinant is close to zero), making the computation of its inverse unstable. So in the following sections we will work with Direct elimination K to ensure the best computation correctness.

We had also provided a figure showing the conditioning of the assembled primal Schur matrix, $\mathbb{S}_p$, varying as a function of $\frac{h}{H}$. But since the figure is very strange that we don't know how to explain, we decided to remove it.

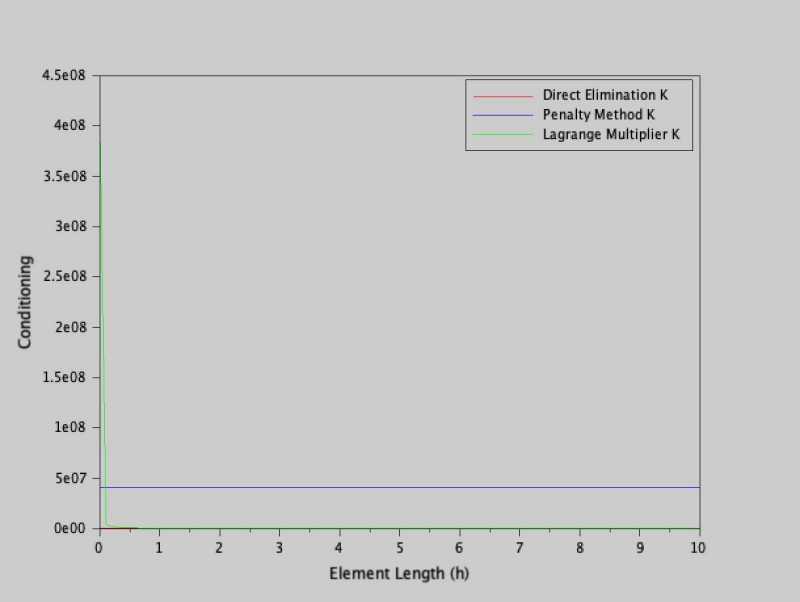

Figure 4: K conditioning according to h

3.2 Distributed conjugate gradient

3.2.1 Implementation

The implementation is done to be as similar as the provided `Algorithm2`, the given initial guess of the displacement is directly included in the beginning of the function in the preliminary part.

3.2.2 Correctness

Please see the blue points in figure2, we can remark that primal iterative resolution also gives relatively low error.

3.2.3 Scalability

However we can see in Fig.5 that this method is not numerically scalable. From Fig3, we observe that the green plots are very dispersed showing a bad parallel scalability too !

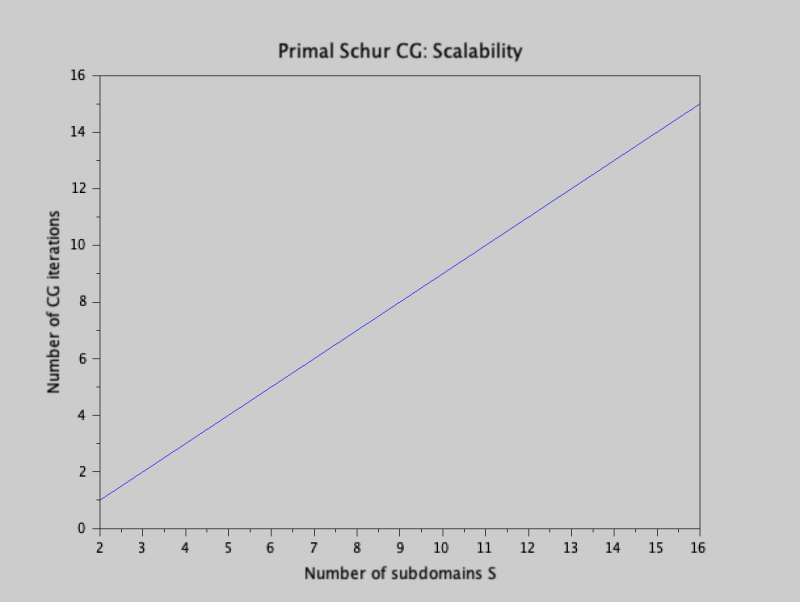

Figure 5: Scalability of Primal CG

3.3 Preconditioned conjugate gradient

3.3.1 Implementation

The implementation is done according to "Iterative solvers for DDM Algo.1", the given initial guess of the displacement u0 is directly included in the beginning of the function the preliminary part. The challenge of the implementation is indeed to identify the possible distributed local operations, in our version the local computation parts include : Sp*u operations and Sp*d operations which contribute to the computation of r and p. To note that in this implementation we simplify some computations, for example the matrix $m^{\diamond}$ for scaled assembly operator $\tilde{\mathbb{A}}$ is not explicitly coded since it's a identity matrix.

3.3.2 Correctness

Please see figure.2. We can see that the errors are slightly higher but compared to Dual approach, the results are still very accurate.

3.3.3 Conditioning

We can see that cond($\tilde{\mathbb{S}}_{|}^{-1}\mathbb{S}_{|}$) ≈ 1, meaning that the system is not ill-conditioned, and can be solve rapidly and correctly. Indeed, we can see that the system is solved at iteration 0, immediately.

3.3.4 Scalability

However, as shown by the concentrated blue points in Fig.3, the execution time of this method is little dependent on variation of the subdomain size H. This fact proves that the preconditioned conjugate gradient version is well parallel scalable.

4 Dual Schur approach

4.1 Direct method

4.1.1 Implementation

The `Dual_direct` function solves for the interface displacement using a dual formulation approach, with a preconditioned CG algorithm, it returns $\alpha_b^\diamond$ such that :

$$\begin{pmatrix} \mathbf{S}_d & \mathbf{G} \\ \mathbf{G}^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda}_b \\ \alpha_b^\diamond \end{pmatrix} = \begin{pmatrix} -b_d \\ -e^\diamond \end{pmatrix}$$

4.1.2 Correctness

We can note that the solutions of this method is very instable and inaccurate, indeed firstly we can see that under some contexts the returned outputs are zero, Fig2 also shows that the direct dual resolution has the highest error. Another remark for the pseudo inverse function "pvin()" used in this case is that this function actually gives different results according to the Mac hardware.

4.2 Preconditioned conjugate gradient

4.2.1 Implementation

The implementation is done according to "Iterative solvers for DDM Algo.2", the given initial guess of lambda0 is directly included in the beginning of the function the preliminary part. Analogical to primal preconditioned conjugate gradient, the local operations include : Sp*lambda and Sd*d. The matrix Q is not explicitly coded since the structure is homogeneous so it's an identity matrix. Note that the post processing phase is also directly implemented into the function.

4.2.2 Correctness

In Fig.2, we can observe that in most of the relative errors is well controlled under 10^{-10}, except the two methods for Dual Schur Complement, which return us a weak reliability of results. As mentioned before, this might be caused by the instability of matrix pseudo inverse computation (`pinv`).

4.2.3 Scalability

in Fig.3, the preconditioned Dual approach is well parallel scalable, since it's runtime does not vary too much with the subdomain size H.

5 Mixed approach

5.0.1 Implementation

The implementation is done according to "Iterative solvers for DDM Algo.4". Note that the stop criterion of the iteration is not computed with η as provided in the paper due to the difficulty of its computation, but the classical "norm()" is used to compare with the tolerance.

5.0.2 Correctness

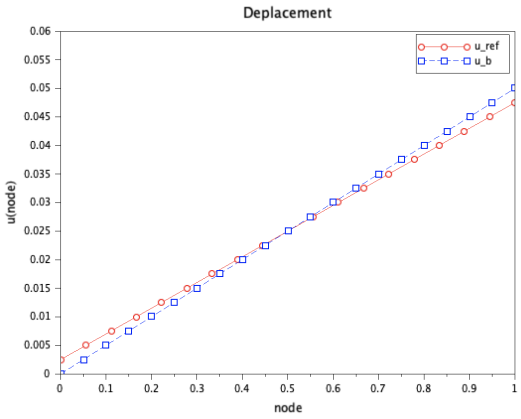

Figure 6: Totally 100 nodes, 5 nodes in each subdomain

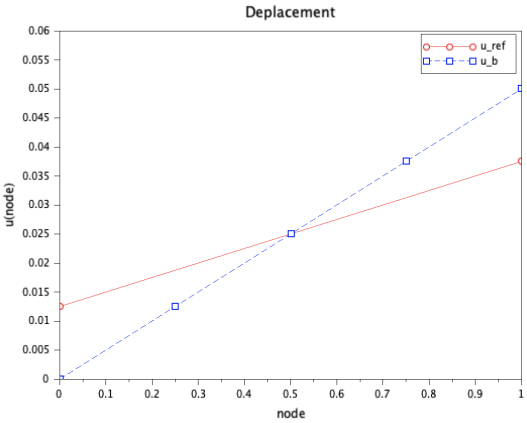

Figure 7: Totally 20 nodes, 5 nodes in each subdomain

We can remark that when the number of substructures increase the results computed by LATIN get closer to the reference solution. As written in the course this approach requires more quantities of interfaces, and require to store and solve these quantities.

5.0.3 Optimal interface stiffness k

After trying different k, we found out that the optimal `inter_k0` in the code is $\frac{EA}{h}\frac{1}{(s-1)}$ where s is the number of the interface. For the 1st interface `inter_k0` is set to a very large number like $1000\frac{EA}{h}$ and the last one is set to 0.

5.0.4 Scalability

We can see that the method is not numerically scalable, which is normal, it's a mono-scale method.

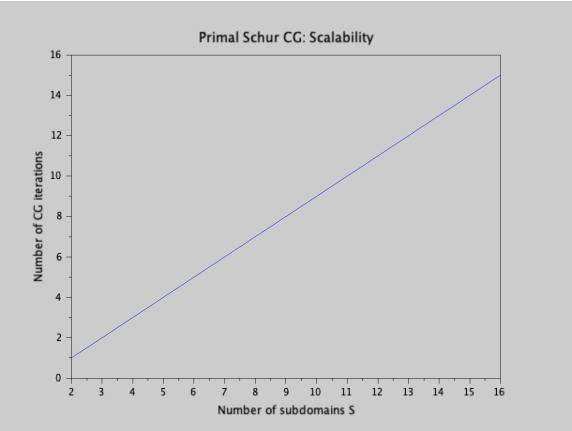

Figure 8: Scalability of LATIN

6 Appendix

The detailed construction process of matrices can be found on github, named "Construction.pdf"