Fast Direct Sparse Solvers implementation

Zhe Chen, Guanchun Li (NYU Courant) *

December 9, 2020

1 Introduction

Elliptic PDEs are of vital significance in applied math. An iterative solver for elliptic PDEs can reach linear or near-to-linear complexity. However, it usually turns unstable as the machine error will accumulate along iterations if the stiffness matrix is ill-conditioned, which is true for large size problem. A direct solver, on the other side, only requires one-time computation but is quite expensive since it requires solving a large linear system. In 2d, a naive dense LU would require $O(N^3)$, and $O(N^2)$ if we take advantage of sparsity of the stiffness matrix. Even if we optimize the ordering of the point to minimize the fill-in of LU and computation cost, such as nested dissection or sweeping scheme, we can only achieve $O(N^{3/2})$ complexity, which works great for moderate size problems but not realistic for large size problem. Fortunately, we notice that the off-diagonal of the Schur complement, which is the main computing cost of a nested dissection scheme, is rank-structured and thus compressible to low rank factorization. This will enable us to build the fast direct sparse solvers (FDSS) based on the compressibility of Schur complement, which is linear or near to linear complexity. In practice, we usually do a FDSS solve quickly, which is 'low accuracy, low cost', and then use it as a preconditioner of the iterative solver to make it fast and stable.

In this final project, we implemented the FDSS (the codes are available in a Github repo) using the sweeping scheme and its acceleration with 'buffering', i.e. buffered sweeping scheme. We test the two schemes in problems of Laplacian or Helmholtz equations and study the influence of parameters on the efficiency of the algorithm and verify its linear complexity. In the end we discuss about some current difficulties and possible future works to improve the performance.

*New York University

2 Sweeping Scheme and Implementation

The sweeping algorithm is a fast direct sparse solver which is easy to describe and to implement, and can achieve linear complexity when taking advantage of the rank-structured algebra. In this section we introduce the idea of the sweeping algorithm and present an implementation in MATLAB. We encourage readers to read Chapter 22 of [1] for more details of the algorithm. But be careful that there are several errors in the pseudo-code in the books, which we found in the process of implementing the algorithm and corrected it in Alg.1.

2.1 Basic sweeping scheme

Given the uniform grid in Fig.1, we can partition the nodes by columns. That's to say, we can separate all nodes $I = I_1 \cup I_2 \cup \cdots \cup I_n$, while each I_k contains the kth column of the grid.

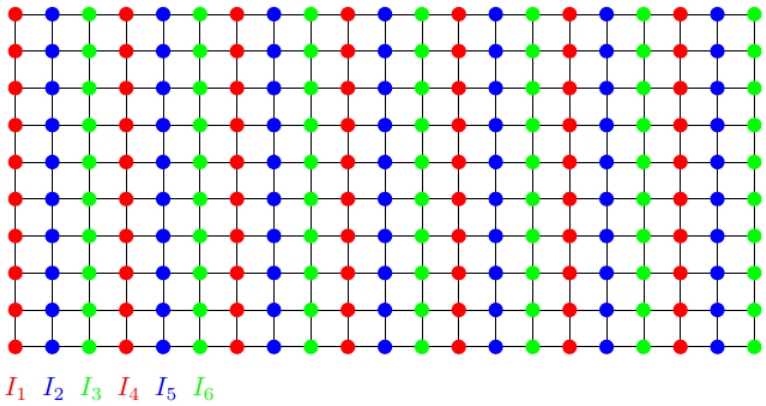

Figure 1: An illustration of the partition of nodes in sweeping scheme.

Given that partition, with suitable discretization scheme (for example, five-point stencil), we will end up with the following block-triangular linear system $Au = f$:

$$\begin{pmatrix} A_{1,1} & A_{1,2} & \mathbf{0} & \mathbf{0} & \cdots \\ A_{2,1} & A_{2,2} & A_{2,3} & \mathbf{0} & \cdots \\ \mathbf{0} & A_{3,2} & A_{3,3} & A_{3,4} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix}. \tag{2.1}$$

Applying the Gaussian elimination with column-by-column ordering, we will end up with algorithm presented in Alg.1.

However, notice that the Schur complements (S_k) in the algorithm are rank structured and hence highly compressible, which can be represented with a ranked structured algebra format. Given this fact, the inversion step ($S_k^{-1}A_{k-1,k}$) can be run in a close to linear time, which greatly accelerates the speed of the whole scheme. More details on the representation of ranked structured algebra will be discussed in section 2.3.

Algorithm 1 A basic sweeping scheme

Build all solution operators in a rightward sweep:
$S_1 = A_{1,1}$
for $k = 2 : m$ **do**
  $S_k = \tilde{A}_{I_k,I_k} - A_{k,k-1}S_{k-1}^{-1}A_{k-1,k}$
end for

Given a load f, compute equivalent loads for the block-tridiagonal system:
$\tilde{f}_1 = f(I_1)$
for $k = 2 : m$ **do**
  $\tilde{f}_k = f(I_k) - A_{k,k-1}S_{k-1}^{-1}\tilde{f}_{k-1}$
end for

Given the equivalent loads, compute solutions in a leftward sweep:
$u(I_m) = S_m^{-1}\tilde{f}_m$
for $k = (m-1) : (-1) : 1$ **do**
  $u(I_k) = S_k^{-1}\left(\tilde{f}_k - A_{k,k+1}u(I_{k+1})\right)$
end for

2.2 Buffered sweeping algorithm

Although theoretically the basic sweeping scheme can achieve a close-to-linear complexity, the significant overhead introduced by the new data structure for the rank structured format makes it not a quite economic choice for the realistic problem sizes. Here we introduce a 'buffered' sweeping algorithm which helps us reduce the size of the linear system while keeping the advantage of the rank structure algebra.

Still given the uniform grid, now the points are partitioned as shown in Fig.2, which are divided into two groups, the 'reduced group' and the 'buffers'. Similar to the idea of the Gaussian elimination, we eliminate the points in the 'buffers' (the blue nodes) to get a reduced linear system for the 'reduced group' points (the red nodes).

The process described above leads us to the algorithm in Alg.2. More details can be found in Chapter 22 of [1] but be careful there are several errors in the pseudo-code in that book. We first induce the reduced linear system $\tilde{A}\tilde{u} = \tilde{f}$ and solving the reduced system with the basic sweeping scheme. Then we take the reduced solution $\tilde{u}$ back to get the full solution u.

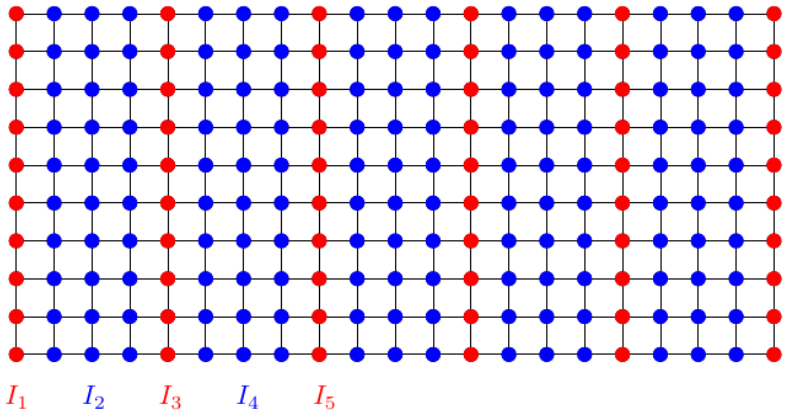

Figure 2: An illustration of the partition of nodes in buffered sweeping scheme.

Notice that for the diagonal blocks $\tilde{A}_{2k-1,2k-1}$ in the reduced system, they still hold the rank structured property, which ensures that it can be accelerated during the sweeping step. Beside, computing $\tilde{A}$ from A is trivially parallelizable since each 'buffer' can be eliminated independently. The parallel implementation would greatly help in the practice for realistic problem size.

2.3 Implementation of the algorithm

In the implementation of the project, we choose MATLAB as the computing platform for the balance between the efficiency and the readability of the programs. Besides, we use the MATLAB package FLAM ([2]) for the conventions of the ranked structure algebra.

Two of the core functions we use in FLAM is 'hifie2' and 'hifie2_sv', while the first is the factorization for the ranker structured matrix and the second is the 'solve' (multiplication of the inverse matrix) given the factorization. Both of them are expected to hold a quasi-linear complexity.

Something to point out is that our codes are still sub-optimal considering we didn't do the parallelization for buffered scheme and we have not realized the full potential of the FLAM package. To be more specific, there are many parameters to play with for the factorization in the ranked structure algebra and we have not fully explored into their affects on the efficiency of the program. It would be something really interesting to see how to choose appropriate parameters for the rank structure algebra given the size and properties of the working problem.

Algorithm 2 A buffered sweeping scheme

Initialize the block-tridiagonal matrix $\tilde{A}$ by copying over the diagonal blocks from A.
for $k = 1 : (m+1)$ **do**
$\tilde{A}_{2k-1,2k-1} = A_{2k-1,2k-1}$.
end for

Eliminate all buffer nodes (the loop can be executed in any order):
for $k = 1 : m$ **do**
$\tilde{A}_{2k-1,2k-1} = \tilde{A}_{2k-1,2k-1} - A_{2k-1,2k} A_{2k,2k}^{-1} A_{2k,2k-1}$
$\tilde{A}_{2k+1,2k+1} = \tilde{A}_{2k+1,2k+1} - A_{2k+1,2k} A_{2k,2k}^{-1} A_{2k,2k+1}$
$\tilde{A}_{2k-1,2k+1} = -A_{2k-1,2k} A_{2k,2k}^{-1} A_{2k,2k+1}$
$\tilde{A}_{2k+1,2k-1} = -A_{2k+1,2k} A_{2k,2k}^{-1} A_{2k,2k-1}$
end for

Factorize the block-tridiagonal system in a rightward sweep:
$S_1 = \tilde{A}_{1,1}$
for $k = 1 : m$ **do**
$S_{2k+1} = \tilde{A}_{2k+1,2k+1} - \tilde{A}_{2k+1,2k-1} S_{2k-1}^{-1} \tilde{A}_{2k-1,2k+1}$
end for

Given a load f, compute equivalent loads for the block-tridiagonal system:
for $k = 1 : (m+1)$ **do**
$\tilde{f} = f(I_{2k-1})$
end for
for $k = 1 : m$ **do**
$\tilde{f}_{2k-1} = \tilde{f}_{2k-1} - A_{2k-1,2k} A_{2k,2k}^{-1} f(I_{2k})$
$\tilde{f}_{2k+1} = \tilde{f}_{2k+1} - A_{2k+1,2k} A_{2k,2k}^{-1} f(I_{2k})$
end for

Given factorization, compute the equivalent loads
for $k = 1 : m$ **do**
$\tilde{f}_{2k+1} = \tilde{f}_{2k+1} - \tilde{A}_{2k+1,2k-1} S_{2k-1}^{-1} \tilde{f}_{2k-1}$
end for

Given the equivalent loads, compute solutions on the interfaces in a rightward sweep:
$u(I_{2m+1}) = S_{2m+1}^{-1} \tilde{f}_{2m+1}$
for $k = m : (-1) : 1$ **do**
$u(I_{2k-1}) = S_{2k-1}^{-1} \left(\tilde{f}_{2k-1} - \tilde{A}_{2k-1,2k+1} u(I_{2k+1}) \right)$
end for

Solve for the solutions on all buffer nodes (the loop can be executed in any order):
for $k = 1 : m$ **do**
$u(I_{2k}) = A_{2k,2k}^{-1} \left(f(I_{2k}) - A_{2k,2k-1} u(I_{2k-1}) - A_{2k,2k+1} u(I_{2k+1}) \right)$
end for

3 Numerical Results

In this part we present some applications of the sweeping scheme with/without buffering on the Laplacian and Helmholtz equations. Influences of the parameters, such as the tolerance, the size of buffer and etc, will also be discussed.

3.1 Laplacian Equation

First we use the basic sweeping algorithm to solve the following Laplacian equation on the square $\Omega = [0,1] \times [0,1]$ in 2D with Dirichlet boundary condition:

$$\begin{cases} -\Delta u = 0, & \forall x \in \Omega \\ u = g(x), & \forall x \in \partial\Omega. \end{cases} \tag{3.1}$$

For the test case here we set BC to be $g = \sin(\pi x)e^{\pi y}$, which leads to the real solution $u = \sin(\pi x)e^{\pi y}$.

Here we investigate how the problem size and tolerance, which is the accuracy to decide the numerical rank of matrix approximation, will affect the performance of the scheme. The results are shown in Fig.3.

We can see that the for given tolerance ϵ, the running time grows with problem size $N = n^2$ almost linearly. That validates our program is actually close to linear complexity. Besides, with smaller tolerance (higher accuracy for matrix approximation), the computation time grows a little, which is quite reasonable since the singular value of the Schur complement decays exponentially.

Remark 1. *Notice that with higher accuracy (tolerance from $1e-1$ to $1e-9$), the growth of computation time is subtle. From the current investigation, we suppose it's due to that the off-diagonal block are highly compressible (the singular value decreases exponentially), hence only few additional steps are needed to get a higher accuracy. But it's not a strict proof yet.*

3.2 Helmholtz Equation

Now we turn to solve the Helmholtz equation (still on the unit square) with Dirichlet boundary condition:

$$\begin{cases} -\Delta u_\kappa = \kappa^2 u, & \forall x \in \Omega \\ u = g_\kappa(x), \forall x \in \partial\Omega. \end{cases} \tag{3.2}$$

For the test case here we set BC to be $g_\kappa = \sin(\kappa x) + \cos(\kappa y)$, which leads to the real solution $u_\kappa = \sin(\kappa x) + \cos(\kappa y)$.

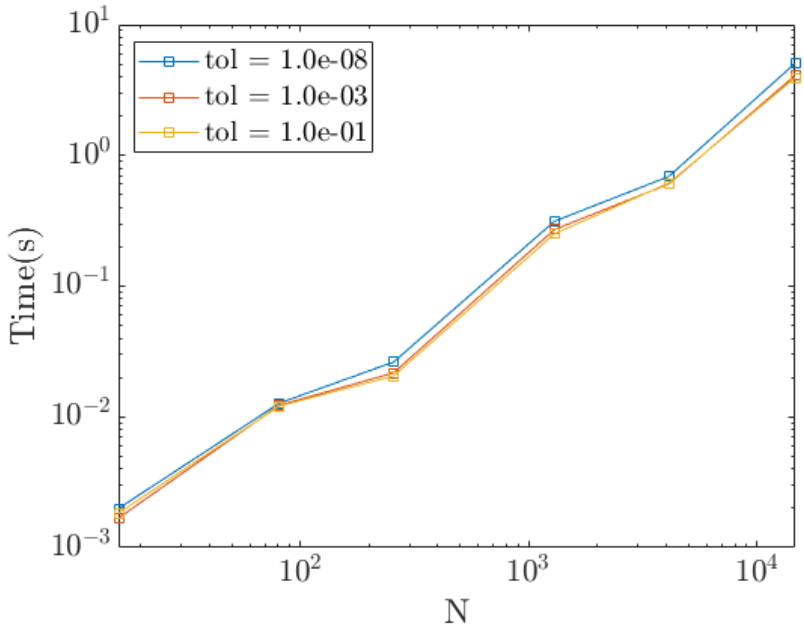

Figure 3: Running time of the basic sweeping scheme on Laplacian equation on 5pt stencil with different problem size N and different tolerance. The algorithm achieves close to linear complexity. The computing time grows a little bit with smaller tolerance.

Here we investigate how the problem size $N = n^2$ and wavenumber κ of Helmholtz equation, which basically tells how oscillatory the solution of Helmhotz equation is, will affect the performance of the scheme. The results are shown in Fig.4.

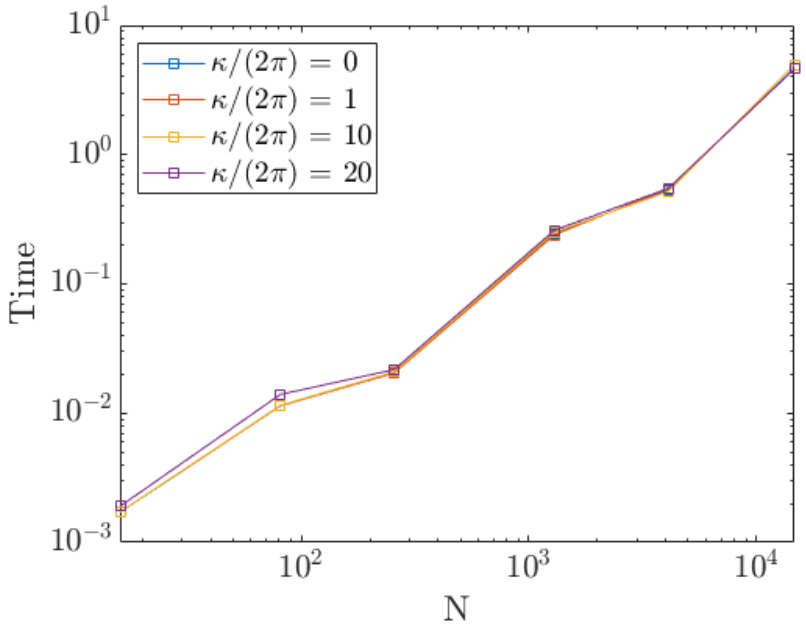

Figure 4: Running time of the basic sweeping scheme on Helmholtz equation with different problem size N and different wavenumber κ. The algorithm achieves close to linear complexity and it's not sensible to the wavenumber.

We can see that the for given wavenumber κ, the running time grows with problem size $N = n^2$ almost linearly, just as what we have seen for the Laplacian equation. Actually, when $\kappa = 0$, it's Laplacian. Besides, the program shows no observable difference for different wavenumbers, which is reasonable since N is much bigger than the wavenumeber κ, indicating that the algorithm could be a suitable choice for oscillatory problems.

3.3 Effects of the 'buffering'

Now we turn to investigate how the 'buffering' improves the performance of the sweeping scheme.

We use the same test case as in Sec.3.1 and fix the grid size to be 169×169. Then we vary to buffer size b from 0 (no buffer) to 83 (only 3 columns survive) and see how the computation time changes. The results are shown in Fig.5.

We can see that with appropriate choice of the buffer size ($b \approx 20, t \approx 0.7\ s$), the buffered sweeping scheme can accelerate a lot (around $10\times$) compared with the non-

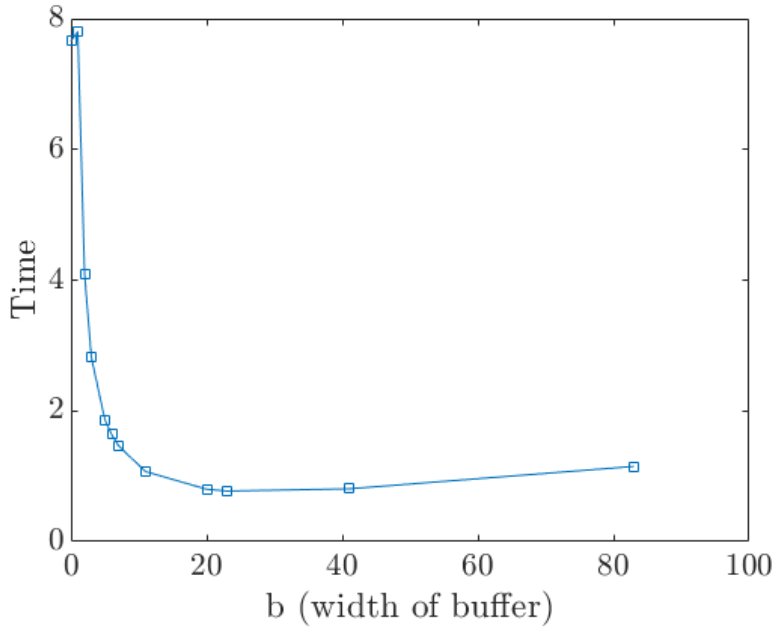

Figure 5: Running time (seconds) versus the width of buffer. Buffered sweeping scheme for Laplace equation with the 5pt stencil. $n_1 = 169$

buffered case ($b = 0, t \approx 7.2\ s$). It's noticeable improvement considering that we haven't even applied the parallelization for computing $\tilde{A}$, which actually can be quite time-consuming when the size is large.

Besides, the buffered scheme achieves the highest efficiency with the modest buffer size b (around 20). It's quite reasonable as it reflects the trade-off between the acceleration of the buffer to the sweeping scheme and the consumption of eliminating the buffers.

4 Conclusion and Discussion

In conclusion, we implement the FDSS algorithm in matlab and achieve linear complexity by using the optimal ordering of LU, i.e. nested dissection, and the suboptimal sweeping scheme, and exploiting the compressibility of the Schur complement. We verify the linear complexity by studying two elliptic PDEs, Laplace's and Helmhotz's equation. Although sweeping scheme may seem not optimal as nested dissection, it is easy to implement and still linear, and moreover preserve the physics. When the problem size is large, especially it's very long in one direction compared to the other, buffered sweeping scheme is better than simple sweeping scheme since it has less overhead and is parallizable for the first

stage.

There are still work that can be done in the future. First, we can also implement the nested dissection scheme though it's more complicated and involved. And the parallelization of the nested dissection and the first stage of the buffered sweeping scheme would also be interesting.

References

[1] MARTINSSON, Per-Gunnar. Fast direct solvers for elliptic PDEs. Society for Industrial and Applied Mathematics, 2019.

[2] K.L. Ho. FLAM: Fast linear algebra in MATLAB – algorithms for hierarchical matrices. J. Open Source Softw. 5 (51): 1906, 2020. doi:10.21105/joss.01906.