Math221 Final Project: Evaluation of Solvers on Frequency-Domain Finite Difference Numerical Modelling of Acoustic Waves

Hussain AlSalem

May 9, 2016

1 Introduction

Seismic exploration is one of the main geophysical methods to extract quantitative inferences about the Earth's interior at different scales from the recording of seismic wave near the surface. Some of the main Applications include civil engineering for cavity detection, nuclear waste storage and oil and gas exploration. Quantitative inference of the physical properties of the subsurface from the recordings of seismic waves at receiver positions is the so-called seismic inverse problem that can be recast in the framework of local numerical optimization. Full waveform inversion (Virieux and Operto (2009) for review) aims to exploit the full information content of seismic data by minimization of the misfit between the full seismic wave-field and the modeled one. The dominance of compressional (P) wave speed relative to shear wave (S) in marine environments has prompted many authors to develop and apply seismic modeling and inversion under the acoustic approximation, either in time domain or frequency domain.

In frequency-domain, wave modeling reduces to the resolution of a complex-valued large and sparse system of linear equations for each frequency, the solution of which is the monochromatic wave-field and the right hand side (RHS) is the source. Three key issues in frequency domain wave modeling concern the linear algebra technique used to solve the linear system, adding variable topography in the large and sparse system of linear equations and the numerical method used for the discretization of the wave equation. In this study, I will thoroughly investigate the first issue.

1.1 Linear algebra solver techniques.

The linear system can be solved with Gauss elimination techniques based on sparse direct solver (Duff et al., 1986), Krylov-subspace iterative methods (Saad, 2003) or hybrid direct/iterative method and domain decomposition techniques (Smith et al., 2004). If the framework of seismic imaging includes a large number of sources (i.e., RHS), one motivation behind the frequency-domain formulation of acoustic wave modeling has been to develop efficient approaches for multi-RHS modeling based on sparse direct solvers. A sparse direct solver first performs an LU decomposition of the matrix which is independent of the source followed by forward and backward substitutions for each source to get the solution (Demmel, 1997). Two drawbacks of the direct-solver approach are the memory requirement of the LU decomposition resulting from the fill-in of the matrix during the LU decomposition and the limited scalability of the LU decomposition on large-scale distributed-memory platforms. In this report, I will compare between two robust solvers: SuperLU direct solver (Li et al., 2011) and IDR(s) which is an efficient iterative solver short recurrence Krylov subspace method for solving large nonsymmetric systems of linear equations (Sonneveld and van Gijzen, 2008). The goal of this report is to show the advantages, drawbacks and limits of these two solvers regarding our forward problem.

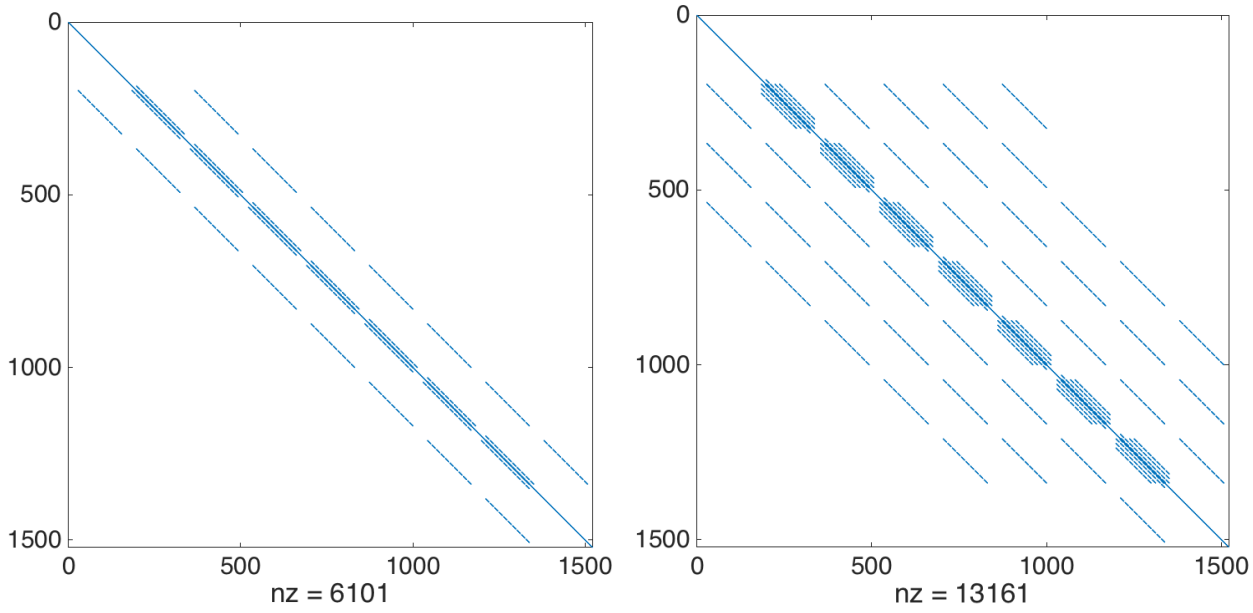
2 Frequency-domain acoustic wave equation

Following standard Fourier transformation convention, the 3D acoustic first-order velocity-pressure system can be written in the frequency domain as:

$$\begin{aligned} -i\omega p(x,y,z,w) &= \kappa(x,y,z)\left(\frac{\partial v_x(x,y,z,\omega)}{\partial x}+\frac{\partial v_y(x,y,z,\omega)}{\partial y}+\frac{\partial v_z(x,y,z,\omega)}{\partial z}\right) \\ -i\omega v_x(x,y,z,\omega) &= b(x,y,z)\cdot\frac{\partial p(x,y,z,\omega)}{x}+f_x(x,y,z,\omega) \\ -i\omega v_y(x,y,z,\omega) &= b(x,y,z)\cdot\frac{\partial p(x,y,z,\omega)}{y}+f_y(x,y,z,\omega) \\ -i\omega v_z(x,y,z,\omega) &= b(x,y,z)\cdot\frac{\partial p(x,y,z,\omega)}{z}+f_z(x,y,z,\omega), \end{aligned} \tag{1}$$

where ω is the angular frequency, $\kappa(x,y,z)$ is the bulk modulus, $b(x,y,z)$ is the buoyancy, $p(x,y,z,\omega)$ is the pressure, $v_x(x,y,z,\omega)$, $v_y(x,y,z,\omega)$, $v_z(x,y,z,\omega)$ are the components of the particle velocity vector. $f_x(x,y,z,\omega)$, $f_y(x,y,z,\omega)$, $f_z(x,y,z,\omega)$ are the components of the external forces. The first block of Equation 1 is the time derivative of the Hooke's law, while the three last block rows are the equation of motion in the frequency domain.

The first-order system can be recast as a second-order equation in pressure after elimination of



(a) 2nd order FD

(b) 4th order FD

Figure 1: 2nd and 4th order discretized matrices $\mathbf{A}$ for a $13 \times 13 \times 9$ that includes boundary conditions.

the particle velocities in Equation 1, that leads to a generalization of the Helmholtz equation:

$$\frac{\omega^2}{\kappa(\mathbf{x})} p(\mathbf{x},\omega) + \frac{\partial}{\partial x} b(\mathbf{x}) \frac{\partial p(\mathbf{x},\omega)}{\partial x} + \frac{\partial}{\partial y} b(\mathbf{x}) \frac{\partial p(\mathbf{x},\omega)}{\partial y} + \frac{\partial}{\partial z} b(\mathbf{x}) \frac{\partial p(\mathbf{x},\omega)}{\partial z} = s(\mathbf{x},\omega), \qquad (2)$$

where $\mathbf{x} = (x, y, z)$ and $s(\mathbf{x},\omega) = \nabla \cdot \mathbf{f}$ denotes the pressure source. Since the relationship between the wavefields and the source terms is linear in the first-order and second-order wave equations, Equations 1 and 2 can be recast in matrix form:

$$[\mathbf{M} + \mathbf{S}]\mathbf{u} = \mathbf{A}\mathbf{u} = \mathbf{b}, \qquad (3)$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{S}$ is the complex stiffness/damping matrix. The sparse impedance matrix $\mathbf{A}$ has complex-valued coefficients which depend on medium properties and angular frequency. The wavefield (either the scalar pressure wavefield or the pressure-velocity wavefields) is denoted by the vector $\mathbf{u}$ and the source by $\mathbf{b}$ (Marfurt, 1984). The dimension of the square matrix $\mathbf{A}$ is the number of pressure nodes in the computational domain. The matrix $\mathbf{A}$ has a symmetric pattern for the 2nd and 4th order finite difference (FD) method discussed in this study but is generally not symmetric because of absorbing boundary conditions along the edges of the computational domain. An example of a 2nd order and 4th order FD discretization of matrix $\mathbf{A}$ for a $13 \times 13 \times 9$ domain are shown in Figure 1.

3 Algorithms of the solvers

3.1 IDR(s) solver

The IDR methods are based on the induced dimension reduction (IDR) method (Sonneveld and van Gijzen, 2008). IDR(s) generates residuals that are forced to be in sequence of nested subspaces. Although IDR(s) behaves like an iterative method, in exact arithmetic it computes the true solution using at most $N + N/s$ matrix-vector products, with N the problem size and s the co-dimension of a fixed subspace. From Sonneveld and van Gijzen (2008), the IDR(s) pseudo-code is shown in Algorithm 1.

3.2 SuperLU solver

The kernel algorithm in SuperLU is sparse Gaussian elimination (Li, 2005). The high-level algorithm can be summarized as follows:

1. Compute a *triangular factorization* $P_r D_r A D_c P_c = LU$. Here D_r and D_c are diagonal matrices to equilibrate the system, P_r and P_c are *permutation matrices*. Premultiplying A by P_r reorders the rows of A, and post-multiplying A by P_c reorders the columns of A. P_r and P_c are chosen to enhance sparsity, numerical stability, and parallelism. L is a unit lower triangular matrix ($L_{ii} = 1$) and U is upper triangular matrix. The factorization can also be applied to non-square matrices.

2. Solve $AX = B$ by evaluating $X = A^{-1}B = (D_r^{-1}P_r^{-1}LUP_c^{-1}D_c^{-1})^{-1}B = D_c(P_c(U^{-1}(L^{-1}(P_r(D_rB)))))$. This is done efficiently by multiplying from right to left in the last expression: Scale the rows of B by D_rB. Multiplying $L^{-1}(P_rD_rB)$ means solving the right hand sides (nRHS) triangular systems of equations with matrix L by substitution. Similarly, multiplying $U^{-1}(L^{-1}(P_rD_rB))$ means solving triangular systems with U.

The pseudo-code for the right-looking algorithm is given by Algorithm 2. The algorithm is chosen for parallel SuperLU for the following reasons:

1. The sparsity structure can be determined before numerical factorization because of static pivoting.

2. All the GEMM (general matrix multiplication) updates to the trailing submatrix are independent and so can be done in parallel.

Algorithm 1 IDR(s) pseudo-code.

Require: $\mathbf{A} \in \mathbb{C}^{N \times N}; \mathbf{x}_0, b \in \mathbb{C}^{N \times s}; TOL \in (0,1); MAXIT > 0$
Ensure: x_n such that $||\mathbf{b} - \mathbf{A}x_n|| \leq TOL$

(Initialization)
Calculate $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$;
(Apply s minimum norm steps, to build enough vectors in $\mathbf{G}_0$)
for $n = 0$ **to** $s - 1$ **do**
    $v = \mathbf{A}\mathbf{r}_n; \omega = (\mathbf{v}^{\mathbf{H}} r_n)/(\mathbf{v}^H \mathbf{v})$;
    $\Delta\mathbf{x}_n = \omega\mathbf{r}_n; \Delta\mathbf{r}_n = -\omega\mathbf{v}$;
    $\mathbf{r}_{n+1} = \mathbf{r}_n + \Delta\mathbf{r}_n; \mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$;
end for
$\Delta\mathbf{R}_{n+1} = (\Delta\mathbf{r}_n \cdots \Delta\mathbf{r}_0); \Delta\mathbf{X}_{n+1} = (\Delta\mathbf{x}_n \cdots \Delta\mathbf{x}_0)$;
(Building $\mathbf{G}_j$ spaces for $j = 1, 2, 3, \cdots$)
$n = s$
(Loop over $\mathbf{G}_j$ spaces)
while $||\mathbf{r}_n|| > TOL$ **and** $n < MAXIT$ **do**
    (Loop inside $\mathbf{G}_j$ space)
    for $k = 0$ **to** s **do**
        Solve $\mathbf{c}$ from $\mathbf{P}^H \Delta\mathbf{R}_n \mathbf{c} = \mathbf{P}^H \mathbf{r}_n$
        $\mathbf{v} = \mathbf{r}_n - \Delta\mathbf{R}_n \mathbf{c}$;
        if $k = 0$ **then**
            (Entering $\mathbf{G}_{j+1}$)
            $\mathbf{t} = \mathbf{A}\mathbf{v}$;
            $\omega = (\mathbf{t}^H \mathbf{v})/(\mathbf{t}^H \mathbf{t})$;
            $\Delta\mathbf{r}_n = -\Delta\mathbf{R}_n \mathbf{c} - \omega\mathbf{t}$;
            $\Delta\mathbf{x}_n = -\Delta\mathbf{X}_n \mathbf{c} + \omega\mathbf{v}$;
        else
            (Subsequent vectors in $\mathbf{G}_{j+1}$)
            $\Delta\mathbf{x}_n = -\Delta\mathbf{X}_n \mathbf{c} + \omega\mathbf{v}$;
            $\Delta\mathbf{r}_n = -\mathbf{A}\Delta\mathbf{x}_n$;
        end if
        $\mathbf{r}_{n+1} = \mathbf{r}_n + \Delta\mathbf{r}_n$;
        $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$;
        $n = n + 1$;
        $\Delta R_n = (\Delta r_{n-1} \cdots \Delta r_{n-s})$;
        $\Delta X_n = (\Delta x_{n-1} \cdots \Delta x_{n-s})$;
    end for
end while

Algorithm 2 SuperLU pseudo-code.

1: **for** block $K = 1$ **to** N **do**
2: Factorize $A(K:N, K) \rightarrow L(K:N, K)$ (may involve pivoting)
3: Compute $U(K, K+1:N)$ (via a sequence of triangular solves)
4: Update $A(K+1:N, K+1:N) \leftarrow A(K+1:N, K+1:N) - L(K+1:N, K) \cdot U(K, K+1:N)$ (via a sequence of calls to GEMM)
5: **end for**

4 Local machine simulations

A 2011 iMac machine was used to conduct tests for small matrix sizes to both quality control the code and find the limits of the machine. The iMac has 3.5GHZ quad-core Intel Core i7 with 4MB on-chip shared L3 cache. It also has 16GB of 1333MHz DDR3 memory. All local machine tests were conducted for a second order FD discretization.

4.1 Free surface boundary (FSB) results

The first simulation was done for a flat surface with FSB. For this simulation, the square matrix $\mathbf{A}$ dimension and number of unknowns is approximately equal to the cubic of the total number of cells in the x direction. The number of unknowns was increased gradually to find the limits of each solver in the local machine. The tested solvers on the local machine are sequential iterative IDR(s), sequential MATLAB direct solver, sequential SuperLU and parallel SuperLU with four processors. Figure 2 shows the elapsed times and limits of these solvers. The Sequential Matlab solver could only solve up to approximately $50K$ (40 cells in each direction) unknowns while sequential SuperLU limit was approximately $300K$ unknowns (67 cells in each direction). On the other hand, sequential IDR(s) and 4-core parallel SuperLU limit was approximately $600K$ (85 cells in each direction) unknowns. For large matrices with $200K$ (60 cells in each direction) and more unknowns, the IDR(s) sequential solver is about 5 times faster than 4-core parallel SuperLU and 10 times faster than sequential SuperLU.

4.2 Staircase boundary results

The second simulation was done for a staircase surface with FSB. In this case, we have a 10-degree inclined surface at the first third part of the domain. All nodes (unknowns) located above this surface have a pressure value equal to zero. For this simulation, the square matrix $\mathbf{A}$ dimension and number of unknowns is approximately equal to the cubic of the total number of cells in the x direction. In addition, the cells above the surface will have a corresponding value equal to one in the diagonal of the $\mathbf{A}$ matrix and zero in the RHS. The number of unknowns was increased

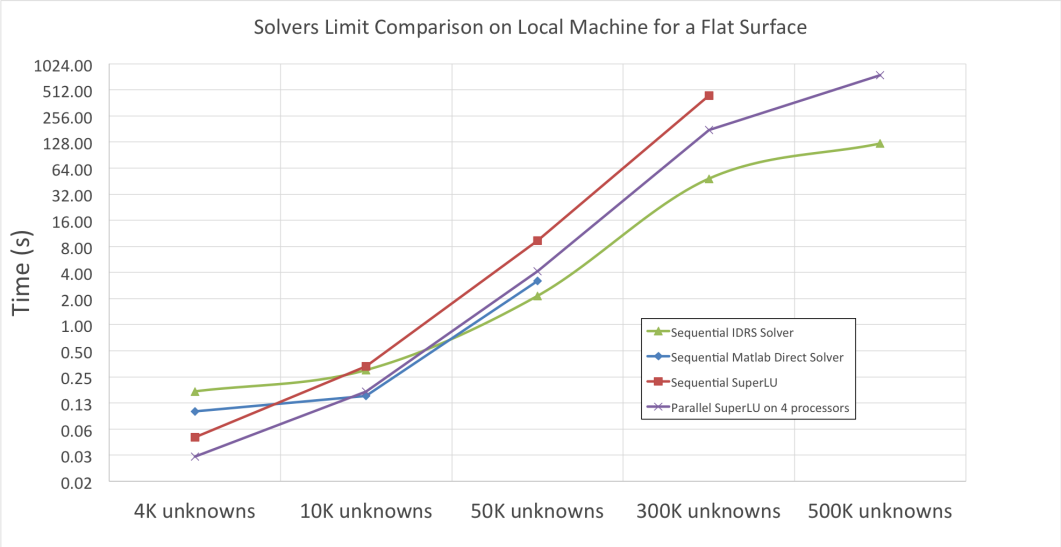


Figure 2: Comparison of speeds and limits of direct solvers on local machine (iMac 2011) with flat topography.

gradually to find the limits of each solver in the local machine. The tested solvers on the local machine were the same as for the flat surface. Figure 3 shows the elapsed times and limits of these solvers. The Sequential Matlab solver could only solve up to approximately $50K$ (40 cells in each direction) unknowns. On the other hand, sequential IDR(s), sequential SuperLU and 4-core parallel SuperLU limit was approximately $600K$ (85 cells in each direction) unknowns. For large matrices with $200K$ (60 cells in each direction) and more unknowns, the IDR(s) sequential solver is about 3 times faster than 4-core parallel SuperLU and 10 times faster than sequential SuperLU.

5 Direct SuperLU vs iterative IDR(s) parallel solvers on Edison

In order to test strong and weak scaling, limits, advantages and drawbacks of each solver on real-sized problems, further tests on NERSC were made. The tests were conducted on Edison which is a Cray XC30 super-computer with peak performance of 2.57 petaflops/sec, 133,824 compute cores, 357 terabytes of memory, and 7.56 petabytes of disk. Each core has about 2.67 gigabytes of memory and a speed of 2.4 GHz. Furthermore, each node has two sockets, each socket is populated with a 12-core Intel "Ivy Bridge" processor, 24 cores per node. The simulations in this section were all performed using a 2.67 gigabytes of memory per core and conducted on a Flat topography problem with FSB.

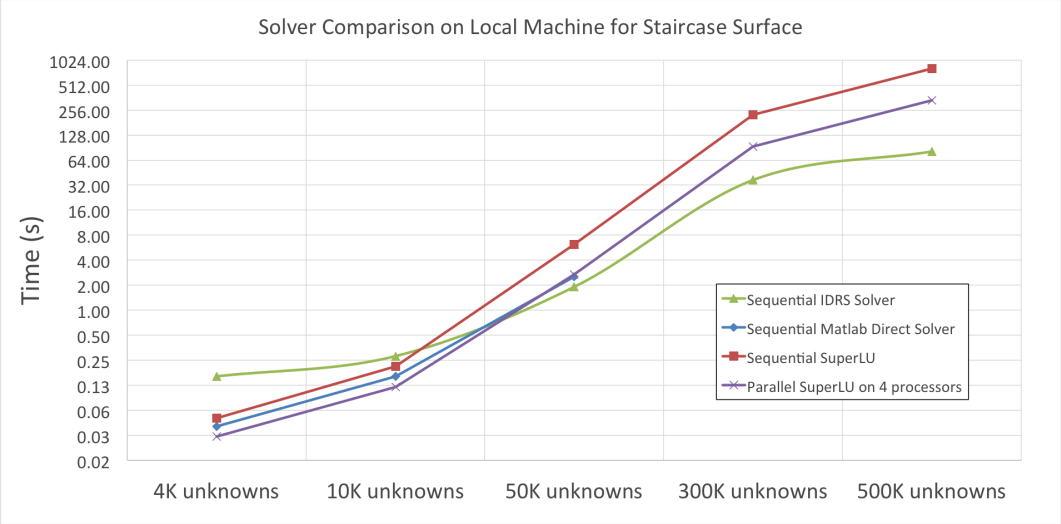


Figure 3: Comparison of speeds and limits of direct solvers on local machine (iMac 2011) with staircase topography.

5.1 Second-order results

The first simulation was conducted to measure the strong scaling for a medium-sized problem. The problem size $\mathbf{A}$ was kept constant and the number of processors increased gradually from 1 to a 100. The results in Figure 4 and Figure 5 show that IDR(s) methods are approximately 10 times faster than parallel SuperLU for mid-sized problems. Thus, SuperLU direct solvers for these problem sizes will be cost-effective when we have more than 10 sources (10 different RHS sides). Furthermore, Figure 6 implies that the SuperLU direct solver scales better than the IDR(s) solver as the number of processors increases.

The second simulation, conducted with the 2nd order discretization, is weak scaling. For this type of scaling, the problem size increases linearly with the number of processors. This test was conducted to find the efficiency and limits of the solvers using fully packed nodes on Edison. From Figure 7, we observe that both the IDR(s) and SuperLU solvers are efficient at weak scaling. SuperLU had a 0.48 slope and IDR(s) had a -0.04 slope. Furthermore, during our tests, SuperLU could not solve larger problems (with more than 3M unknowns) on packed nodes as it runs out of memory. This means that the SuperLU solver is limited to second order mid-sized (approximately 3M unknowns) problems when using 2.67GB of memory per node.

5.2 Fourth-order weak-scaling results

Results from Section 5.1 led to further tests in order to find the fill-in memory limits of SuperLU when solving a 4th order discretization problem. Weak scaling was again used to find the limits of SuperLU for the 4th order FD scheme. Figure 8 shows that the SuperLU solver could only

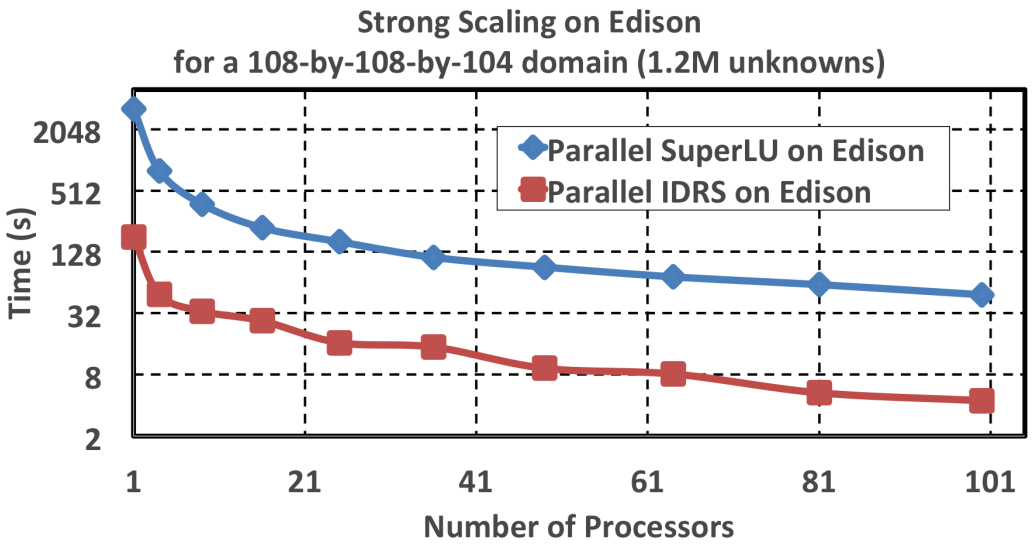


Figure 4: Strong-scaling comparison on Edison for SuperLU and IDR(s) solvers using 2nd order discretization scheme.

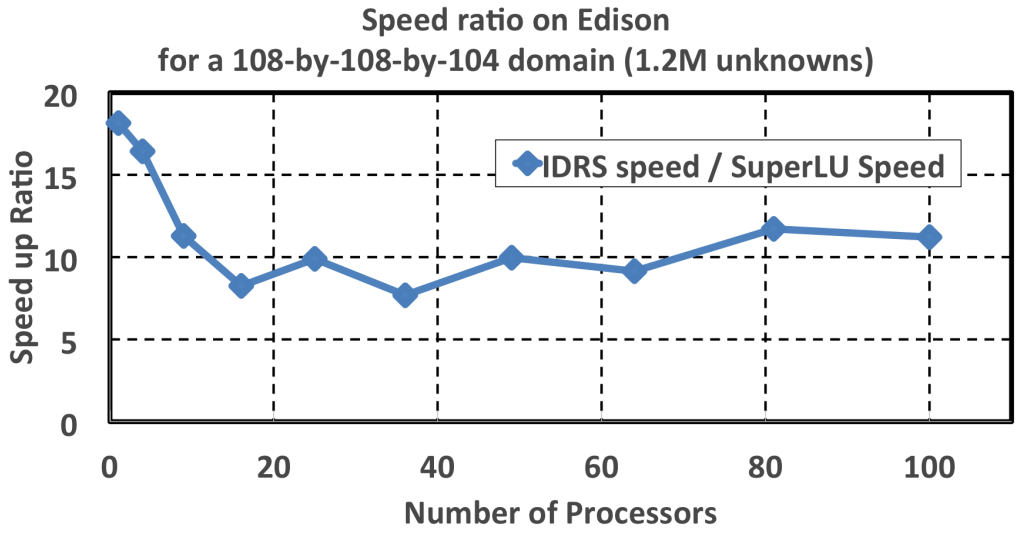


Figure 5: Speed ratio comparison on Edison for IDR(s)÷SuperLU solvers using 2nd order discretization scheme.

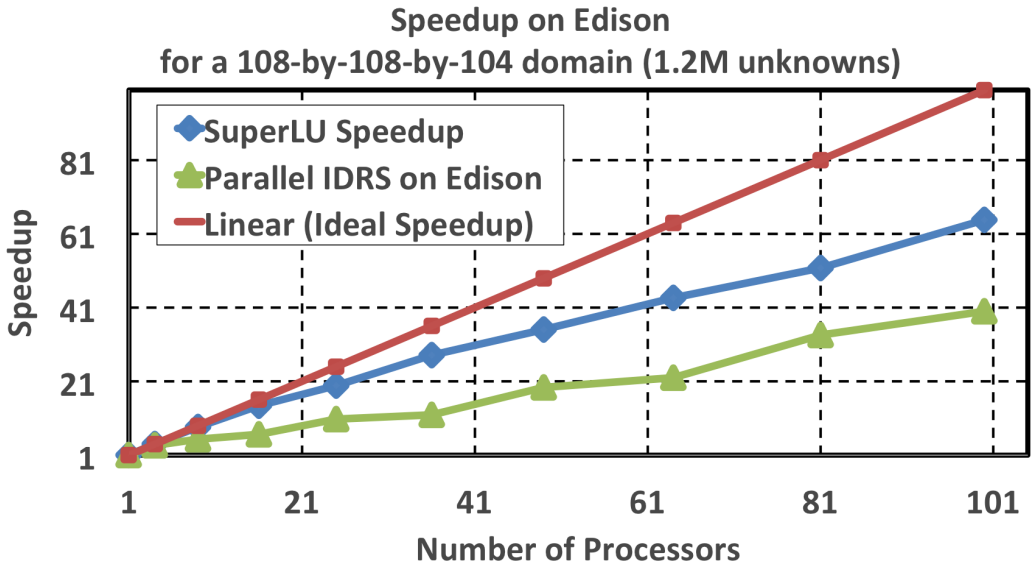


Figure 6: Speedup comparison on Edison for IDR(s) and SuperLU solvers using 2nd order discretization scheme.

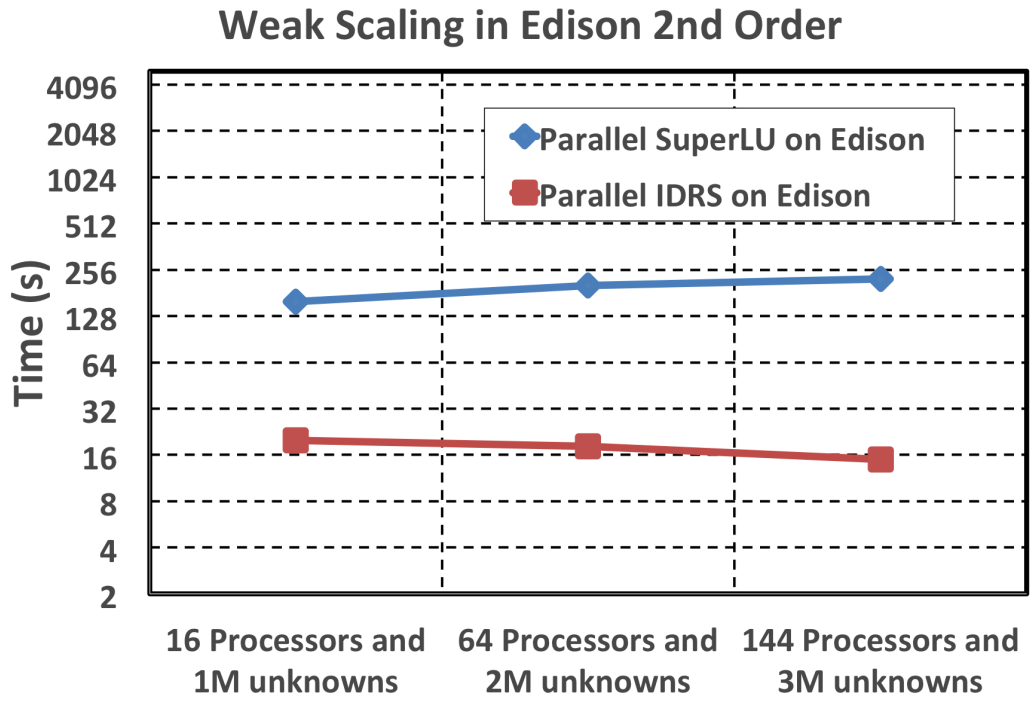


Figure 7: Weak-scaling comparison on Edison for SuperLU and IDR(s) solvers using 2nd order discretization scheme.

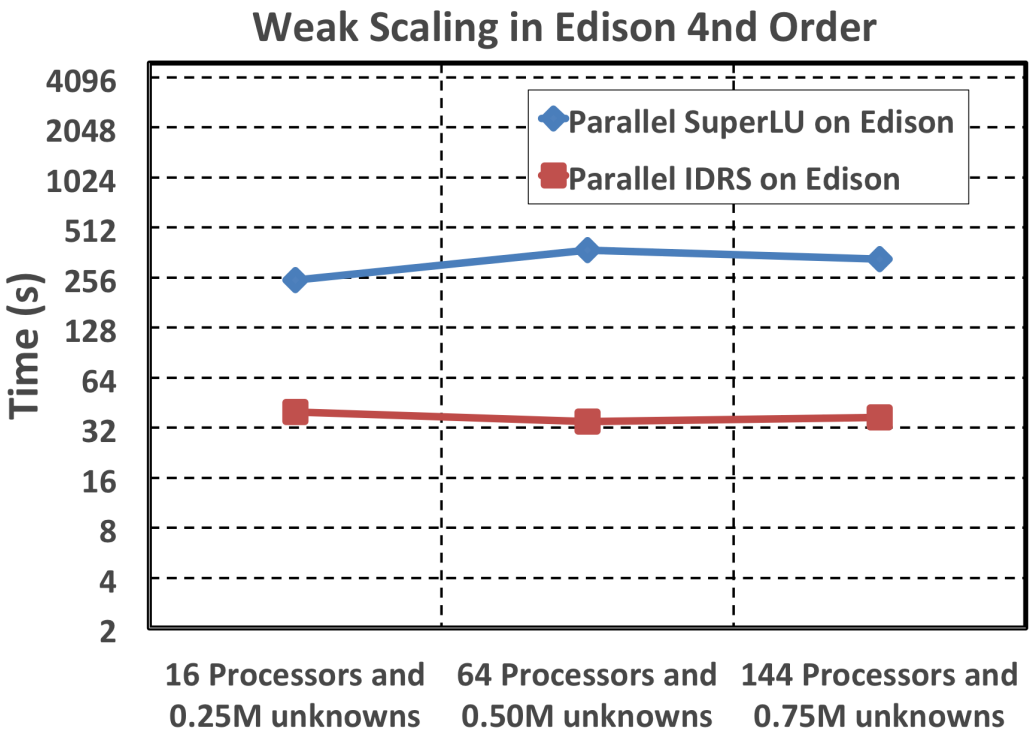


Figure 8: Weak-scaling comparison on Edison for SuperLU and IDR(s) solvers using 4th order discretization scheme.

solve small-sized problems with less than 100 cells in each direction (1M unknowns). We also observe that both the IDR(s) and SuperLU solvers are efficient at weak scaling. SuperLU had a 0.52 slope and IDR(s) had a -0.01 slope. Therefore, for fourth order discretization, SuperLU solver is limited to small-sized (less than 1M unknowns) problems when using 2.67GB of memory per node on Edison.

6 Unpacked nodes results for second-order scheme

The last comparison was done on Edison to solve large matrices that could not be solved on packed nodes due to out of memory (OOM) errors. On the packed nodes results, all processors were used for both computation and memory storage leaving 2.67 Gigabytes (GB) per core for the SuperLU solver. On the other hand, the unpacked nodes test only uses two cores for computation and the rest are left for memory storage. This leaves 32 GB of memory per core. Figure 9 shows the performance of SuperLU compared to the IDR(s) solver. Since the IDR(s) solver does not run out of memory, it will use all the cores for computation. This makes the IDR(s) solver much faster as the problem size gets bigger. For example, from Figure 9, we can see that it takes 120 sources (RHS sides) for SuperLU to break even with IDR(s) when solving a problem with 6 Million unknowns.

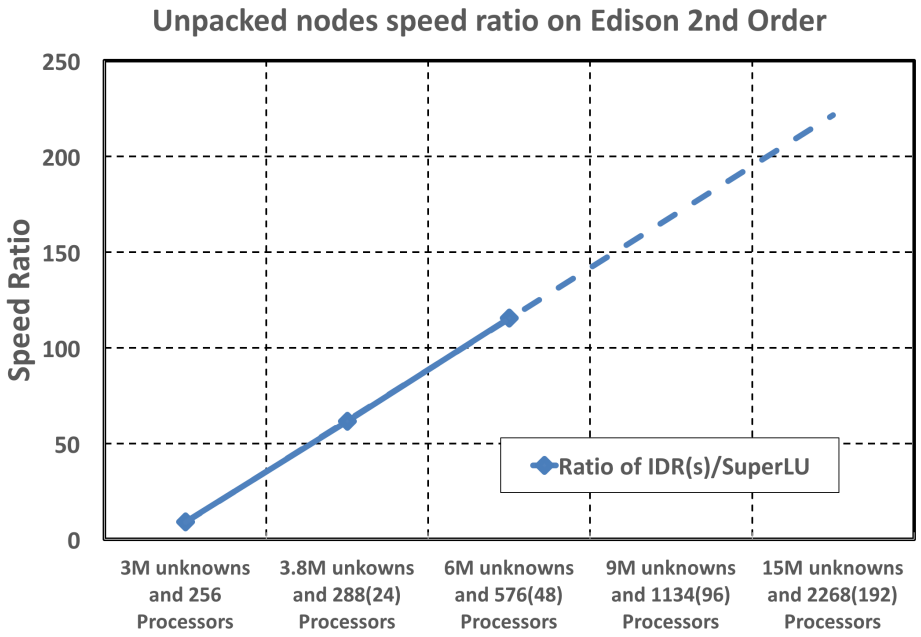


Figure 9: Comparison of the speed ratio when unpacked nodes are used for large problems. The numbers between parenthesis are the computational processors used for SuperLU. The rest of the processors are not used to conserve memory.

7 Conclusion

The results show that for the second order scheme, small-sized to mid-sized matrices ($100 \times 100 \times 100$ domain with 1 million unknowns to $200 \times 200 \times 100$ domain with 4 million unknowns) should be solved using SuperLU instead of IDR(s). Our usual seismic exploration problem will approximately have 100 sources. Since SuperLU is just ten times slower than IDR(s), it will be faster to factorize matrix A and solve for the multiple RHS using SuperLU. Generally, for this problem size, the forward and backward substitutions take less than 1 second. Thus, making the IDR(s) method inferior. However, for large-size problems ($200 \times 200 \times 200$ domain with 6 million unknowns and more), the current implementation of parallel SuperLU is more than 100 times slower than IDR(s). Thus, making IDR(s) method superior. Part of the future work will include tweaking the parallel SuperLU algorithm to make it better than IDR(s) for even large-sized problems.

Future work The future goals of this project will be to cheaply run SuperLU by using less CPU hours and less memory. The first task, as discussed with *Prof. Demmel*, is to use nested dissection ordering. Nested dissection is a method of finding elimination ordering. The algorithm uses a *divide and conquer strategy* on the graph. Removal of a set of vertices results in two new graphs which Gaussian elimination may be performed separately. The results for the two parts may then

be combined to find the solution of the entire graph. This method will be tested for our matrices as it has been shown to result in good elimination orderings of certain cases similar to ours (Khaira et al., 1992). The second task will be to perform these simulations again in Cori as it has 32 cores and 128GB of memory per node. This configuration will hugely decrease CPU hours drain compared to Edison's 24 cores and 64GB of memory per node.

References

Demmel, J. W. (1997). *Applied numerical linear algebra*. Siam.

Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). *Direct methods for sparse matrices*. Clarendon press Oxford.

Khaira, M. S., Miller, G. L., and Sheffler, T. J. (1992). *Nested Dissection: A survey and comparison of various nested dissection algorithms*. Carnegie-Mellon University. Department of Computer Science.

Li, X. S. (2005). An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325.

Li, X. S., Demmel, J. W., Gilbert, J. R., Grigori, L., Shao, M., and Yamazaki, I. (2011). Superlu users' guide. *Internet Address: http://crd. lbl. gov/˜ xiaoye/SuperLU/superlu_ug. pdf*.

Marfurt, K. J. (1984). Accuracy of finite-difference and finite-element modeling of the scalar and elastic wave equations. *Geophysics*, 49(5):533–549.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. Siam.

Smith, B., Bjorstad, P., and Gropp, W. (2004). *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge university press.

Sonneveld, P. and van Gijzen, M. B. (2008). Idr (s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM Journal on Scientific Computing*, 31(2):1035–1062.

Virieux, J. and Operto, S. (2009). An overview of full-waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC1–WCC26.