Computers & Geosciences 61 (2013) 184–197

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

An optimized parallel LSQR algorithm for seismic tomography

En-Jui Lee [a,*], He Huang [b], John M. Dennis [c], Po Chen [a], Liqiang Wang [b]

[a] Department of Geology and Geophysics, University of Wyoming, Dept. 3006, 1000 E University Ave, Laramie, WY 82071, United States
[b] Department of Computer Science, University of Wyoming, Dept. 3315, 1000 E University Ave, Laramie, WY 82071, United States
[c] Computational Science Section, Scientific Computing Division, National Center for Atmospheric Research, 1850 Table Mesa Drive Boulder, CO 80303, United States

ARTICLE INFO

Article history:
Received 23 May 2013
Received in revised form
8 August 2013
Accepted 30 August 2013
Available online 12 September 2013

Keywords:
LSQR algorithm
Tomographic inversion
MPI
Computational seismology
Inverse problems
Parallel scientific computing

ABSTRACT

The LSQR algorithm developed by Paige and Saunders (1982) is considered one of the most efficient and stable methods for solving large, sparse, and ill-posed linear (or linearized) systems. In seismic tomography, the LSQR method has been widely used in solving linearized inversion problems. As the amount of seismic observations increase and tomographic techniques advance, the size of inversion problems can grow accordingly. Currently, a few parallel LSQR solvers are presented or available for solving large problems on supercomputers, but the scalabilities are generally weak because of the significant communication cost among processors. In this paper, we present the details of our optimizations on the LSQR code for, but not limited to, seismic tomographic inversions. The optimizations we have implemented to our LSQR code include: reordering the damping matrix to reduce its bandwidth for simplifying the communication pattern and reducing the amount of communication during calculations; adopting sparse matrix storage formats for efficiently storing and partitioning matrices; using the MPI I/O functions to parallelize the date reading and result writing processes; providing different data partition strategies for efficiently using computational resources. A large seismic tomographic inversion problem, the full-3D waveform tomography for Southern California, is used to explain the details of our optimizations and examine the performance on Yellowstone supercomputer at the NCAR-Wyoming Supercomputing Center (NWSC). The results showed that the required wall time of our code for the same inversion problem is much less than that of the LSQR solver from the PETSc library (Balay et al., 1997).

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

Seismic waves generated by natural or manmade sources and recorded by seismometers carry important information about the physical properties of the subsurface earth structures through which they propagate. Seismic tomography is an imaging technique that assimilates ground-motion observations collected using seismometers to improve structural models of the Earth's interior and it has been one of most effective means for imaging the Earth's interior in the past few decades.

The seismic tomography problem is often formulated as an optimization problem, in which we search for an optimal earth structure model that minimizes an objective function defined in terms of certain misfit measurements that quantify the discrepancies between the observed wave-fields and the corresponding synthetic wave-fields predicted using a reference earth structure model. A typical objective function that is often employed in practice has the quadratic form

$$\chi^2(\mathbf{m}) = \mathbf{d}^T\mathbf{C}_d^{-1}\mathbf{d} + (\mathbf{m}-\tilde{\mathbf{m}})^T\mathbf{C}_m^{-1}(\mathbf{m}-\tilde{\mathbf{m}}), \tag{1}$$

where $\mathbf{d}$ is a vector composed of individual misfit measurements, $\mathbf{m}$ is a vector composed of model parameters, $\tilde{\mathbf{m}}$ is a vector of the reference structure model, $\mathbf{C}_m$ is the *a priori* model covariance matrix and $\mathbf{C}_d$ is the data covariance matrix. This type of objective functions arises in the context of statistical inference based on a Gaussian-Bayesian point of view (e.g., Tarantola, 2005). For an individual misfit measurement d_{in}^{sr}, which is the n-th misfit measurement on the i-th component seismogram generated by source s and recorded at receiver r, the data sensitivity kernel $K_{d_{in}^{sr}}^{\mathbf{m}}(\tilde{\mathbf{m}},\mathbf{x})$ is the functional (Fréchet) derivative of this misfit measurement with respect to the model parameters around the reference model (Backus and Gilbert, 1968), i.e.,

$$\delta d_{in}^{sr} = \int dV(\mathbf{x}) K_{d_{in}^{sr}}^{\mathbf{m}}(\tilde{\mathbf{m}},\mathbf{x})\delta\mathbf{m}(\mathbf{x}). \tag{2}$$

If discretized over space $\mathbf{x}$, the data sensitivity kernel $K_{d_{in}^{sr}}^{\mathbf{m}}(\tilde{\mathbf{m}},\mathbf{x})$ becomes a vector and the spatial integral in Eq. (2) can be expressed as an inner product. The Jacobian matrix $\mathbf{A}_k$ is the

* Corresponding author. Tel.: +1 307 460 1744.
E-mail addresses: elee8@uwyo.edu, rickli92@gmail.com (E.-J. Lee).

0098-3004/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cageo.2013.08.013

matrix with each row given by the discretized data sensitivity kernel for each individual misfit measurement. For nonlinear least-squares problems such as the one defined in Eq. (1), the Gauss–Newton algorithm is often an effective optimization algorithm because the Jacobian matrix, which only involves the first-order Fréchet derivative of every misfit measurement, can provide not only the gradient of the objective function but also an approximation of its Hessian. The exact Hessian of the objective function in Eq. (1) is given by

$$\mathbf{H} = \mathbf{A}_k^T \mathbf{C}_d^{-1} \mathbf{A}_k + \mathbf{C}_m^{-1} + (\nabla_{\mathbf{m}} \mathbf{A}_k)^T \mathbf{C}_d^{-1} \mathbf{d}, \tag{3}$$

which involves the derivative of the Jacobian matrix, therefore second-order derivatives of individual misfits. However, when $\mathbf{d}$ is small and/or the individual misfits are approximately linear with respect to model parameters (i.e., $\nabla_{\mathbf{m}} \mathbf{A}_k$ is small), the last term in Eq. (3) can be neglected. Under such an approximation, if we expand the objective function in Taylor series around the reference model $\tilde{\mathbf{m}}$, truncate the expansion to second-order and then set the derivative of the truncated series with respect to $\mathbf{m}$ to zero, we arrive at the Gauss–Newton normal equation

$$(\mathbf{A}_k^T \mathbf{C}_d^{-1} \mathbf{A}_k + \mathbf{C}_m^{-1})(\mathbf{m} - \tilde{\mathbf{m}}) = \mathbf{A}_k^T \mathbf{C}_d^{-1} \mathbf{d}. \tag{4}$$

In practice, we do not need to explicitly form this equation, because its solution can be computed by solving the linear system

$$\begin{bmatrix} \mathbf{C}_d^{-1/2} \mathbf{A}_k \\ \mathbf{C}_m^{-1/2} \end{bmatrix} (\mathbf{m} - \tilde{\mathbf{m}}) = \begin{bmatrix} \mathbf{C}_d^{-1/2} \mathbf{d} \\ 0 \end{bmatrix} \tag{5}$$

via a relaxation method. The LSQR algorithm of Paige and Saunders (1982) is one of the most popular solvers used in seismic tomography due to its efficiency and stability in solving large, sparse and ill-conditioned linear systems (e.g., Nolet, 1985; Nolet, 1993). Once Eq. (5) is solved, the structure model can be updated and the updated model can become the new reference model for the next iteration. This process can then be iterated until convergence.

In conventional ray-theoretic travel-time tomography, an individual misfit measurement is determined by the difference between the observed travel-time of a specific seismic phase and the corresponding model-predicted travel-time computed using a ray-tracing algorithm in the reference structure model and the data sensitivity kernel is determined by the ray-path connecting the source and the receiver for the selected seismic phase (e.g., Červený, 2005). The "finite-frequency" effect of wave-propagation can be accounted for by combining the Born approximation with the paraxial ray theory and the corresponding data sensitivity kernel exhibits the counterintuitive "banana-doughnut" phenomena, i.e., the sensitivity of the cross-correlation delay-time is non-zero within a tube surrounding the ray path (i.e., the Fresnel zone) but is zero on the ray path (Marquering et al., 1999; Dahlen et al., 2000; Hung et al., 2000; Zhao et al., 2000). Recent advances in parallel computing technology and numerical methods (e.g., Olsen, 1994; Graves, 1996; Bao et al., 1998; Komatitsch and Vilotte, 1998; Komatitsch et al., 2004; Dumbser et al., 2007) have significantly reduced the computational cost for solving acoustic and (visco) elastic seismic wave equations in realistic 3D earth structure models, which has opened up the possibilities for wave-equation-based (i.e., "full-wave") seismic tomography techniques. The adjoint-state method, which was adopted to solve seismic imaging problems in Bamberger et al. (1977, 1982) and later extended to 2D acoustic (Pratt and Worthington, 1990; Pratt et al., 1998) and 3D acoustic and elastic full-wave inversions (e.g., Tarantola, 1984; 1988; Tromp et al., 2004), is numerically efficient for computing the gradient of the objective function, as it only requires one forward and one adjoint wave-propagation simulation per seismic source. For a dataset with N_s seismic sources, the total number of wave-propagation simulations (forward and adjoint) needed for constructing the gradient is $2N_s$. Once the gradient of the objective function is available, gradient-based optimization algorithms such as the steepest-descent and the conjugate-gradient methods can be adopted to minimize the objective function. However the adjoint method is not efficient for constructing the Jacobian matrix, as it will need one forward and one adjoint simulation to compute the data sensitivity kernel for each misfit measurement. For realistic seismic tomography problems involving a large number of misfit measurements, the number of simulations and the computational cost needed to construct the Jacobian matrix using the adjoint method is prohibitive.

The scattering-integral (SI) method (Zhao et al., 2005), which is physically equivalent to, but computationally different from the adjoint method (Chen et al., 2007a), provides a computationally viable approach for constructing the Jacobian matrix. Consider the data sensitivity kernel of the misfit measurement d_{in}^{sr} with respect to the elastic moduli $c_{jklm}(\mathbf{x})$, after applying the reciprocity principle (Aki and Richards, 2002), the data sensitivity kernel can be expressed as (Chen et al., 2007a)

$$K_{d_{in}^{sr}}^{cjklm}(\mathbf{x}) = -\int dt \int d\tau\, J_{in}^{sr}(t) \partial_k G_{ji}(\mathbf{x}, t-\tau; \mathbf{x}_r) \partial_l u_m^s(\mathbf{x}, \tau), \tag{6}$$

where ∂_k represents the partial derivative with respect to x_k, $J_{in}^{sr}(t)$ is the functional derivative of the misfit measurement with respect to the waveform, i.e., $\delta d_{in}^{sr} = \int J_{in}^{sr}(t) \delta u_i^s(\mathbf{x}_r, t)\, dt$, $G_{ji}(\mathbf{x}, t; \mathbf{x}_r)$ is the Green's tensor for a unit impulsive force acting at the receiver location $\mathbf{x}_r$ and is named the "receiver Green's tensor" (RGT), $u_m^s(\mathbf{x}, t)$ is the m-th component forward wave-field generated by the seismic source s. The SI method is based on the observation that the RGTs do not depend on the sources. If we compute and stored them on disk, they can be re-used for constructing the data sensitivity kernels of different seismic sources. The computational cost for carrying out the temporal convolution and integration in Eq. (6) is almost negligible compared to the cost for carrying out a wave-propagation simulation. For a dataset with N_r receivers, the total number of simulations needed to construct the Jacobian matrix using the SI method is $N_s + 3N_r$ for 3D elastic problems. For acoustic and/or 2D problems, the number of simulation is even less. For realistic seismic tomography applications the disk space needed for storing the RGTs can be substantial but still manageable by adopting efficient data compression algorithms.

The SI method has been successfully applied to image the crustal structure of the Los Angeles Basin area in Chen et al. (2007b) and the tomographic inversion is currently being extended to Southern California. The linear system in Eq. (5) is more than 450 times larger in the Southern California inversion than that in the Los Angeles Basin inversion. In Chen et al. (2007b), the LSQR code used for solving Eq. (5) came from the PETSC library (Balay et al., 1997). But when we try to apply the same code to our inversion in Southern California, it does not provide satisfactory performance. In this paper we discuss our optimization of the parallel LSQR algorithm and demonstrate the performance of our code using one Gauss–Newton iteration from our Southern California tomographic inversion. In our Southern California inversion, we have completed 5 Gauss-Newton iterations so far. In each iteration, the updated model from the previous iteration is used as our reference model for the current iteration and the Jacobian matrix is re-computed for the current iteration using the SI method. Our optimized LSQR code is used to solve the Gauss-Newton normal equation in every iteration. The full inversion process based on both the adjoint and the SI method will be documented in a separate publication.

a

LSQR Algorithm

(1. Initialize)

$\beta_1\mathbf{u}_1 = \mathbf{d},\ \alpha_1\mathbf{v}_1 = \mathbf{A}\mathbf{u}_1,\ \mathbf{w}_1 = \mathbf{v}_1,$

$\delta\mathbf{m}_1 = 0,\ \bar{\phi}_1 = \beta_1,\ \bar{\rho}_1 = \alpha_1,\ i = 1;$

while (Converge criterion is false){

(2. Bidiagonalization)

$(a)\ \beta_{i+1}\mathbf{u}_{i+1} = \mathbf{A}\mathbf{v}_i - \alpha_i\mathbf{u}_i;$

$(b)\ \alpha_{i+1}\mathbf{v}_{i+1} = \mathbf{A}^T\mathbf{u}_{i+1} - \beta_{i+1}\mathbf{v}_i;$

(3. Othhogonal transform)

$\rho_i = \sqrt{\bar{\rho}_i^2 + \beta_{i+1}^2},\ c_i = \bar{\rho}_i / \rho_i,\ s_i = \beta_{i+1} / \rho_i,$

$\theta_{i+1} = s_i\alpha_{i+1},\ \bar{\rho}_{i+1} = -c_i\alpha_{i+1},\ \phi_i = c_i\bar{\phi}_i,\ \bar{\phi}_{i+1} = s_i\bar{\phi}_i;$

(4. Update results)

$\delta\mathbf{m}_i = \delta\mathbf{m}_{i-1} + (\phi_i / \rho_i)\mathbf{w}_i;$

$\mathbf{w}_i = \mathbf{v}_{i+1} - (\theta_{i+1} / \rho_i)\mathbf{w}_i;$

$i{+}{+};$

}

b

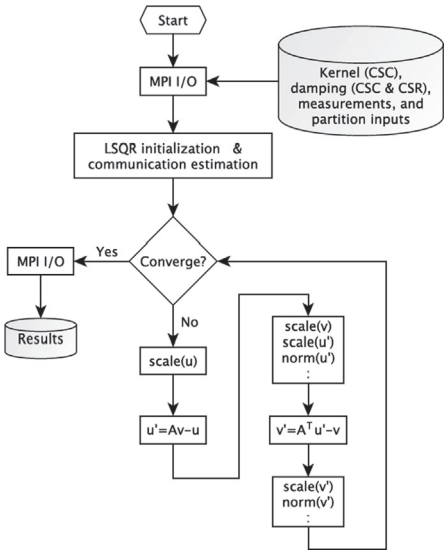


Fig. 1. (a) Main steps in the LSQR algorithm (Paige and Saunders, 1982) and (b) workflow of our implementations. In the workflow, the scale( ) and norm( ) functions represent the scalar-vector multiplication and vector normalization operations.

This paper is organized as follows. In Section 1, we briefly discussed the seismic tomography problem and introduced the LSQR algorithm for solving the optimization problem. An overview of full-wave tomography and the SI method used to construct the Jacobian matrix in our inversion problem is also presented in Section 1. In Section 2, we present the ideas of our optimizations of the LSQR algorithm in detail. The performance analysis of our code, performance comparisons with the LSQR solver in the PETSc and a single iteration result of our Southern California tomographic inversion are presented in Section 3. We conclude our optimizations of the LSQR algorithm and discuss some future developments in the Section 4.

2. Optimization

The main steps in the LSQR algorithm (Paige and Saunders, 1982) and workflow of our implementations are summarized in Fig. 1. In realistic seismic tomography applications, more than 90% of the total computing time is spent on the matrix–vector multiplications **Av** in step (2a) and $\mathbf{A}^T\mathbf{u}$ in step (2b). Here we have dropped the iteration index i for convenience. Our effort has been focused on accelerating these two matrix–vector multiplications by taking advantage of the special structure of the **A** matrix in seismic tomography problems.

As shown in Eq. (5), the matrix **A** is composed of two submatrices (Fig. 2): the Jacobian (or Fréchet) matrix $\mathbf{A}_k$, which is composed of the partial derivatives of the misfit measurements in vector **d** with respect to model parameters in vector **x**, which corresponds to $\mathbf{m}-\tilde{\mathbf{m}}$ in Eq. (5), and the regularization matrix $\mathbf{A}_d$, which corresponds to $\mathbf{C}_m^{-1/2}$ in Eq. (5) and usually contains the identity matrix, for penalizing the norm of the solution vector, and/or finite-difference discretization of the first- or second-order spatial derivatives, for penalizing the spatial roughness of the solution vector. In realistic seismic tomography applications, both $\mathbf{A}_k$ and $\mathbf{A}_d$ are highly sparse and more than 99% of all non-zero elements in matrix **A** are located in the Jacobian matrix $\mathbf{A}_k$. The regularization matrix $\mathbf{A}_d$ can be converted into a band matrix form with a relatively small bandwidth (i.e., for a given row, the number of columns between the first and the last non-zero elements on that row) by permuting the rows and/or columns of the matrix **A**. We will discuss the re-ordering algorithm for matrix $\mathbf{A}_d$ in Section 2.4.

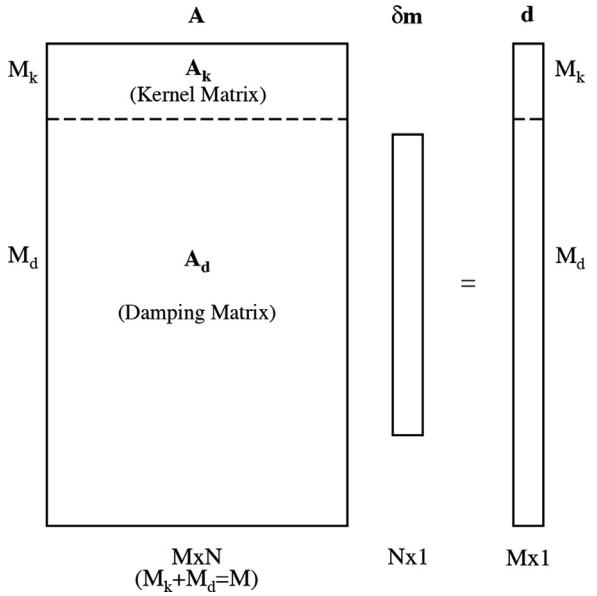


Fig. 2. An illustration of the damped least-squares problem of seismic tomographic inversions. The matrix **A** is composed by the kernel matrix $\mathbf{A}_k$ and the damping matrix $\mathbf{A}_d$, the elements in vector $\delta\mathbf{m}$ are the model perturbations and the elements in the vector **d** that correspond to the kernel matrix are measurements and that correspond to the damping matrix are all zeros.

2.1. Parallel partition of matrices and vectors

On distributed-memory parallel computers, each processor stores only a portion of the matrices and vectors in its own memory and accesses to the portions lying on other processors are usually implemented through inter-processor message passing, which has a higher latency than accesses to the processor's own memory. When we partition the matrices and vectors among a group of processors, we need to balance the computational load on each processor while minimizing the inter-processor communication overhead.

In our implementation, each processor owns a range of columns of the Jacobian matrix $\mathbf{A}_k$ (Fig. 3a). Suppose the Jacobian matrix $\mathbf{A}_k$ has M_k rows and N columns, and then processor p owns the sub-matrix

$$\mathbf{A}_k^p = \mathbf{A}_k(1:M_k, c_b^p : c_e^p), 1 \le c_b^p \le c_e^p \le N \quad (7)$$

and the beginning and ending column indices for processor p, c_b^p and c_e^p, are selected based on load-balanced considerations (Section 2.5) for different processors.

Suppose the regularization matrix $\mathbf{A}_d$ has M_d rows. On each processor, we keep two sub-matrices of $\mathbf{A}_d$. One sub-matrix $\mathbf{A}_d^p$ is composed of a subset of the rows of $\mathbf{A}_d$ (Fig. 3a),

$$\mathbf{A}_d^p = \mathbf{A}_d(r_b^p : r_e^p, 1:N), 1 \le r_b^p \le r_e^p \le M_d \quad (8)$$

and the beginning and ending row indices r_b^p and r_e^p are determined by the bandwidth of the re-ordered regularization matrix $\mathbf{A}_d$ and the column indices c_b^p andc_e^p. The other sub-matrix $\mathbf{A'}_d^p$ is composed of a subset of the columns of $\mathbf{A}_d$ (Fig. 3b),

$$\mathbf{A'}_d^p = \mathbf{A}_d(1:M_d, c_b^p : c_e^p). \quad (9)$$

Since $\mathbf{A}_d$ is highly sparse, both $\mathbf{A}_d^p$ and $\mathbf{A'}_d^p$ are highly sparse too. The procedure for selecting appropriate values for c_b^p, c_e^p, r_b^p and r_e^p in a load-balanced way is described in Section 2.5.

As for the vectors involved in the two matrix–vector multiplications, we introduce two new vectors $\mathbf{u}' = \mathbf{A}\mathbf{v}$ and $\mathbf{v}' = \mathbf{A}^T\mathbf{u}$. The vector $\mathbf{u}$ can be computed from $\mathbf{u}'$ through step (2a) in Fig. 1a and the vector $\mathbf{v}$ can be computed from $\mathbf{v}'$ through step (2b) in Fig. 1a. Processor p owns a portion of the $\mathbf{v}$ vector,

$$\mathbf{v}^p = \mathbf{v}(c_b^p : c_e^p), \quad (10)$$

and also the same portion of the $\mathbf{v}'$ vector. The $\mathbf{u}'$ vector has $M_k + M_d$ elements, the sub-vector $\mathbf{u}'_k$ is composed of the first M_k elements of $\mathbf{u}'$ and the sub-vector $\mathbf{u}'_d$ is composed of the next M_d elements of $\mathbf{u}'$ (Fig. 3a). Each processor has a duplicated copy of the entire $\mathbf{u}'_k$ vector and a portion of the $\mathbf{u}'_d$ vector,

$$\mathbf{u'}_d^p = \mathbf{u}'_d(r_b^p : r_e^p). \quad (11)$$

For vector $\mathbf{u}$, the sub-vector containing the first M_k elements is denoted as $\mathbf{u}_k$, and each processor owns a separate copy of the entire $\mathbf{u}_k$. The sub-vector containing the next M_d elements is denoted as $\mathbf{u}_d$ and is partitioned in the same way as the $\mathbf{u}'_d$ vector.

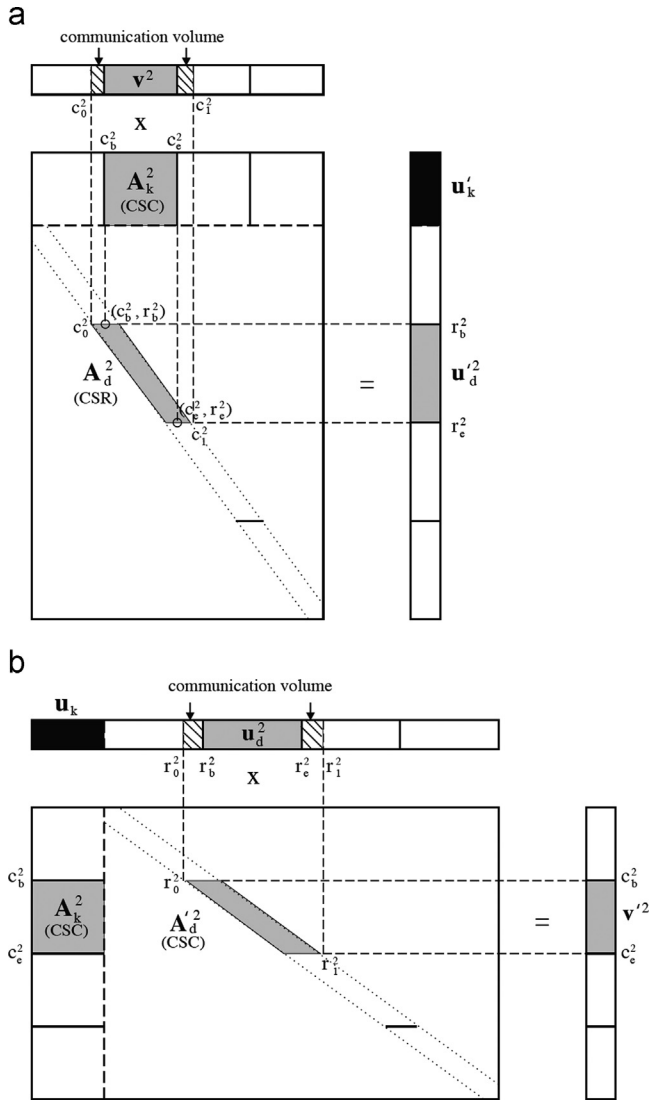


Fig. 3. Illustrations of matrix and vector partitions for four processors and the ideas of the matrix–vector multiplication of (a) $\mathbf{Av}=\mathbf{u}'$ and (b) $\mathbf{A}^T\mathbf{u}=\mathbf{v}'$. The elements that need to be stored in the second processor are in gray and the elements in $\mathbf{u}'_k$ (or $\mathbf{u}_k$) that each processor has a duplicate is in black. During the matrix–vector multiplications of $\mathbf{A}_d^2\mathbf{v}$ and $\mathbf{A'}_d^2\mathbf{u}$, communication among processors are required for completing the operations. The required communication volume from other processors is in strip-pattern.

2.2. Inter-processor communication overhead

The inter-processor communication overhead can be estimated based on the data partition scheme introduced in the previous section. The matrix–vector multiplication $\mathbf{Av}$ (Fig. 3a) can be separated into two steps: the matrix–vector multiplication $\mathbf{A}_k\mathbf{v}$, which generates the vector $\mathbf{u}'_k$, and the matrix–vector multiplication $\mathbf{A}_d\mathbf{v}$, which generates the vector $\mathbf{u}'_d$. Suppose the total number of processors used in the calculation is N_p, since each processor owns only a subset of the columns of $\mathbf{A}_k$, using the index notation we have,

$$\mathbf{u}'_k = \mathbf{A}_k\mathbf{v} = \sum_{j=1}^{N} \mathbf{A}_k(i,j)\mathbf{v}(j) = \sum_{p=1}^{N_p} \left[\sum_{j=c_b^p}^{c_e^p} \mathbf{A}_k^p(i,j)\mathbf{v}^p(j) \right],$$
$$\text{for } i = 1,2,\ldots,M_k, p = 1,2,\ldots,N_p. \quad (12)$$

The term in the square bracket can be computed on each processor without any inter-processor communication and the resulting vector on each processor has M_k elements. The summation over processor index p requires a gather operation to the master processor. If we estimate the communication overhead using the number of "point-to-point" data transfers multiplied with the amount of the data being transferred, the gather operation introduces a communication cost proportional to$(N_p-1)M_k$. Since each processor requires a separate copy of the entire $\mathbf{u}'_k$, we need to broadcast the $\mathbf{u}'_k$ computed on the master processor to all other processors, which introduces a communication cost proportional to $(N_p-1)M_k$. The total amount of inter-processor communication overhead for the matrix–vector multiplication $\mathbf{A}_k\mathbf{v}$ is therefore proportional to around $2(N_p-1)M_k$.

A different partition scheme for $\mathbf{A}_k$ that has been used in some previous studies (e.g. Balay et al., 1997; Huang et al., 2012; Liu et al., 2006) is to partition $\mathbf{A}_k$ along rows, i.e., each processor owns a subset of the rows of $\mathbf{A}_k$. For such a partition scheme, the communication overhead for the matrix–vector multiplication $\mathbf{A}_k\mathbf{v}$ is dependent upon how the vector $\mathbf{v}$ is partitioned. In realistic seismic tomography applications, the vector $\mathbf{v}$ is usually dense and the length of $\mathbf{v}$, which is N, is so large that it is impractical for each processor to store a separate copy of the entire $\mathbf{v}$ vector in its own

memory. If the $\mathbf{v}$ vector is partitioned evenly among all processors, i.e., each processor owns N/N_p elements of $\mathbf{v}$, the communication cost for the matrix–vector multiplication $\mathbf{A}_k\mathbf{v}$ is proportional to around$(N_p-1)N$. For most seismic tomography problems, the linear system is highly under-determined, which means $N \gg M_k$. For the full-wave seismic tomography problem analyzed in this study, N is about 300 times larger than M_k. Considering the communication overhead for our column-based partition scheme, which is proportional to $2(N_p-1)M_k$, for such highly under-determined linear systems, a column based partition schemes may significantly reduce inter-processor communication overhead for the matrix–vector multiplication $\mathbf{A}_k\mathbf{v}$.

The inter-processor communication overhead for the matrix–vector multiplication $\mathbf{A}_d\mathbf{v}$ depends upon the bandwidth of the matrix $\mathbf{A}_d$. For processor p, suppose the first non-zero element on row r_b^p is located at column c_0^p and the last non-zero element on row r_e^p is located at column c_1^p, then we need to transfer the sub-vectors $\mathbf{v}(c_0^p : c_b^p)$ and $\mathbf{v}(c_e^p : c_1^p)$ from other processors to processor p in order for processor p to complete the calculation of $\mathbf{u}'^p_d$ (Fig. 3a). If $c_0^p = c_b^p$ and $c_e^p = c_1^p$ (i.e., the bandwidth of $\mathbf{A}_d$ is one), there is no communication between processor p and other processors. In order to reduce the inter-processor communication overhead for the matrix–vector multiplication $\mathbf{A}_d\mathbf{v}$, we need to minimize the bandwidth of the matrix $\mathbf{A}_d$. In Section 2.4 we show a simple algorithm for reducing the bandwidth of $\mathbf{A}_d$ by permuting the rows of $\mathbf{A}_d$. Suppose the sub-vectors $\mathbf{v}(c_0^p : c_b^p)$ and $\mathbf{v}(c_e^p : c_1^p)$ are distributed among P different processors. Depending upon the dimension of the linear system and the bandwidth of the re-ordered $\mathbf{A}_d$, P can be quite large. However, in practice because the band itself is also highly sparse and we do not need to transfer the elements in $\mathbf{v}$ whose corresponding multiplicands in the band are zeros, the number of processors actually involved in the communication with processor p is usually much less than P. And since the non-zero pattern of $\mathbf{A}_d$ does not change through the LSQR iterations, the processors that need to communicate with processor p, as well as the elements in $\mathbf{v}$ that need to be transferred, can all be identified and registered into a memory buffer prior to the first iteration (Fig. 1b).

The matrix–vector multiplication $\mathbf{A}^T\mathbf{u}$ can be separated into two components (Fig. 3b). Using the index notation, we have

$$\mathbf{v}'(j) = \sum_{i=1}^{M_k+M_d} \mathbf{A}(i,j)\mathbf{u}(i) = \sum_{i=1}^{M_k} \mathbf{A}_k(i,j)\mathbf{u}_k(i) + \sum_{i=1}^{M_d} \mathbf{A}_d(i,j)\mathbf{u}_d(i), \quad \text{for } j = 1,2,\ldots,N. \tag{13}$$

On processor p, we have

$$\mathbf{v}'^p(j) = \sum_{i=1}^{M_k} \mathbf{A}_k^p(i,j)\mathbf{u}_k(i) + \sum_{i=1}^{M_d} \mathbf{A}'^p_d(i,j)\mathbf{u}_d(i), \quad \text{for } j = c_b^p, c_b^p+1, \ldots, c_e^p. \tag{14}$$

Since each processor owns a separate copy of the entire $\mathbf{u}_k$ vector, there is no inter-processor communication in computing the first term on the right-hand-side. Suppose the first non-zero element of $\mathbf{A}_d$ on column c_b^p is located at row r_0^p and the last non-zero element of $\mathbf{A}_d$ on column c_e^p is located at row r_1^p, in order to compute the second term on the right-hand-side, we need to transfer $\mathbf{u}_d(r_0^p : r_b^p)$ and $\mathbf{u}_d(r_e^p : r_1^p)$ from other processors to processor p. By reducing the bandwidth of the regularization matrix $\mathbf{A}_d$ we can also reduce the inter-processor communication cost for the matrix–vector multiplication $\mathbf{A}^T\mathbf{u}$.

2.3. Data structures for matrices

Both the Jacobian matrix $\mathbf{A}_k$ and the regularization matrix $\mathbf{A}_d$ are sparse matrices. The sparseness of the Jacobian matrix $\mathbf{A}_k$ depends both upon the spatial density of the seismic sources and receivers used in the tomography and upon the particular technique used for computing the partial (Fréchet) derivatives of the misfit measurements with respect to the model parameters. In general, for a given seismic source and receiver distribution, $\mathbf{A}_k$ is sparser for ray-theoretical tomography than for finite-frequency or full-wave tomography. For the full-wave tomography analyzed in this study, $\mathbf{A}_k$ has a fill-in ratio of around 3% and $\mathbf{A}_d$ has a fill-in ratio of about 8.2e-6%.

Two of the most widely used data structures for representing sparse matrices are the compressed-sparse-column (CSC) and the compressed-sparse-row (CSR) formats (e.g. Bai et al., 2000). On each processor, the Jacobian sub-matrix $\mathbf{A}_k^p$ is represented using the CSC format, the regularization sub-matrix $\mathbf{A}_d^p$ is represented using the CSR format and the regularization sub-matrix $\mathbf{A}'^p_d$ is represented using the CSC format. Efficient algorithms for matrix–vector multiplications based on both CSC and CSR formats have been developed previously (e.g. Bai et al., 2000).

The benefit of the specific choice of the data structure for each matrix is two-folded: first, it simplifies the procedure for determining the optimal values for c_b^p, c_e^p, r_b^p and r_e^p for each processor; second, it improves the overall throughput when loading the matrices from the disk and partitioning them to the memory of each processor. The procedure for determining c_b^p, c_e^p, r_b^p and r_e^p is discussed in Section 2.5. Once the optimal values for c_b^p, c_e^p, r_b^p and r_e^p are determined, the memory on each processor for storing each sub-matrix in the chosen data format is allocated. On the disk, the entire Jacobian matrix $\mathbf{A}_k$ is stored in a single binary file in the CSC format. We keep two copies of the entire regularization matrix $\mathbf{A}_d$ on the disk, one is stored in the CSR format and the other is stored in the CSC format. Both files are binary. The values for c_b^p, c_e^p, r_b^p and r_e^p can be translated into position pointers and offset values in those binary files and those position pointers and offset values are then used in calling parallel-I/O subroutines, which are the MPI-I/O subroutines in our implementation. The process of reading the data from the disk, partitioning the data among all processors and converting the data into the right format that can be used for subsequent calculations is completed in a single step.

Using the Jacobian matrix $\mathbf{A}_k$ as an example, since it is stored in CSC format, its binary file on the disk is composed of 3 one-dimensional arrays, val, row_ind and col_ptr, where val is an array of the non-zero values in $\mathbf{A}_k$, row_ind is the row indices corresponding to the non-zero values in val and col_ptr stores the indices of the elements in val which start a new column of $\mathbf{A}_k$. For processor p, the sub-matrix $\mathbf{A}_k^p$ is represented using 3 one-dimensional arrays in the memory, val_p, row_ind_p and col_ptr_p. The starting position in the binary file for reading in the val and row_ind arrays for the $\mathbf{A}_k^p$ sub-matrix can be determined from col_ptr(c_b^p) and the offset of the reading operation can be determined from col_ptr(c_e^p)-col_ptr(c_b^p)+1. The sub-arrays read from the val and row_ind arrays on disk fill up the corresponding CSC arrays for representing the sub-matrix $\mathbf{A}_k^p$ in the memory of processor p, i.e.,

$$\text{val\_p} = \text{val}(\text{col\_ptr}(c_b^p) : \text{col\_ptr}(c_e^p)),$$

$$\text{row\_ind\_p} = \text{row\_ind}(\text{col\_ptr}(c_b^p) : \text{col\_ptr}(c_e^p)).$$

The indices in the column-pointer array needs to be adjusted and we have

$$\text{col\_ptr\_p} = \text{col\_ptr}(c_b^p : c_e^p) - \text{col\_ptr}(c_b^p) + 1.$$

A similar approach is also applied to process the two binary files for storing the regularization matrix $\mathbf{A}_d$. In particular, the sub-matrix $\mathbf{A}_d^p$ is extracted from the binary file with the CSR format and the sub-matrix $\mathbf{A}'^p_d$ is extracted from the binary file with the CSC format.

For the full-wave tomography problem analyzed in this study, the size of the binary file for storing the Jacobian matrix $\mathbf{A}_k$ is around 2.1 TB. On the IBM iDataPlex (code named Yellowstone) at the NCAR-Wyoming Supercomputing Center (NWSC), we were able to achieve a sustained I/O rate of 40–50 GB/s. The number of processors used in our experiments ranged from 4000 to 12,000. The theoretical peak I/O rate on the Yellowstone system is around 90 GB/s.

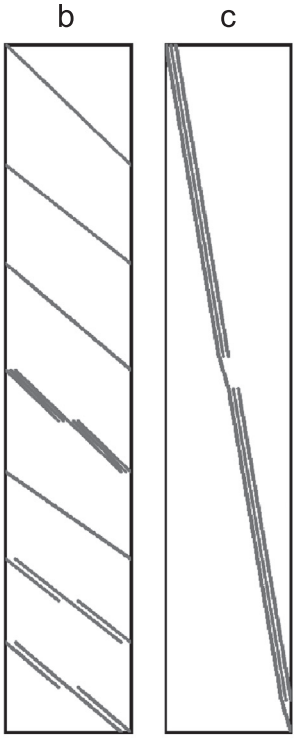
2.4. Re-ordering of the regularization matrix

The structure of the regularization matrix $\mathbf{A}_d$ is determined by the spatial discretization of the solution function $\delta m(\mathbf{x})$ and the regularization operator $\mathbf{D}$. On a rectangular domain discretized using a uniform Cartesian grid, after a simple row-based re-ordering (Fig. 4), finite-difference approximations to the spatial

a

```
row=1
(0th-order)
for iz=1,nz
  for iy=1,ny
    for ix=1,nx
      col=ix+(iy-1)*nx+(iz-1)*nx*ny
      damping[row][col]=1
      row++
    end
  end
end
(∂²/∂x²)
for iz=1,nz
  for iy=1,ny
    for ix=1,nx-2
      col=ix+(iy-1)*nx+(iz-1)*nx*ny
      damping[row][col]=1
      row++
      col=(ix+1)+(iy-1)*nx+(iz-1)*nx*ny
      damping[row][col]=-2
      row++
      col=(ix+2)+(iy-1)*nx+(iz-1)*nx*ny
      damping[row][col]=1
      row++
    end
  end
end
(∂²/∂y²)
...
(∂²/∂z²)
…
(∂²/∂x∂y)
…
(∂²/∂x∂z)
…
(∂²/∂y∂z)
…
```

```
row=1
for iz=1,nz
  for iy=1,ny
    for ix=1,nx
      (0th-order)
      col=ix+(iy-1)*nx+(iz-1)*nx*ny
      damping[row][col]=1
      row++
      if(ix<(nx-2))
        (∂²/∂x²)
        col=ix+(iy-1)*nx+(iz-1)*nx*ny
        damping[row][col]=1
        row++
        col=(ix+1)+(iy-1)*nx+(iz-1)*nx*ny
        damping[row][col]=-2
        row++
        col=(ix+2)+(iy-1)*nx+(iz-1)*nx*ny
        damping[row][col]=1
        row++
      if(iy<(ny-2))
        (∂²/∂y²)
        ...
      if(iz<(nz-2))
        (∂²/∂z²)
        …
      if(ix<(nx-2) & iy<(ny-2))
        (∂²/∂x∂y)
        …
      if(ix<(nx-2) & iz<(nz-2))
        (∂²/∂x∂z)
        …
      if(iy<(ny-2) & iz<(nz-2))
        (∂²/∂y∂z)
        …
    end
  end
end
```



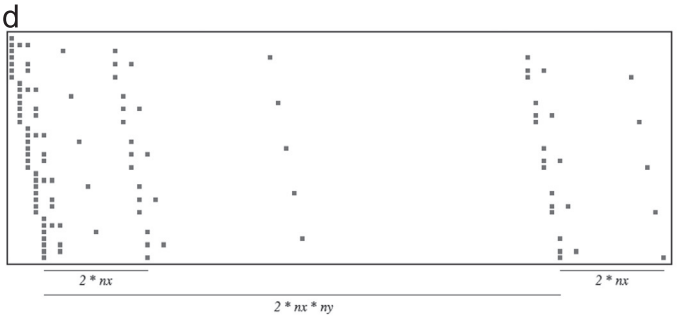


Fig. 4. (a) In the left-hand-side, the original damping matrix, the identity and Laplacian operations, is generated by 7 nested-loops. In the right-hand-side, we reorder the damping matrix by using one nested-loops and conditional statements. (b) The patterns of original damping matrix and (c) our re-ordered damping matrix used in our Southern California full-3D waveform tomographic inversion. (d) The first 35 rows of the re-ordered damping matrix. The gray squares represent non-zero elements.

derivatives result in a banded regularization matrix $\mathbf{A}_d$ and the band itself is also highly sparse.

Consider a rectangular domain discretized into n_x, n_y, and n_z grid points in the x, y and z directions respectively. The three-dimensional grid index (ix,iy,iz), where $ix=1,2,\ldots,n_x$; $iy=1,2,\ldots,n_y$ and $iz=1,2,\ldots,n_z$, can be mapped into a one-dimensional index l using a simple formula,

$$l = ix+(iy-1)n_x+(iz-1)n_xn_y. \tag{15}$$

The three-dimensional solution function $\delta m(\mathbf{x})$ and the Fréchet kernel for each misfit measurement $k(\mathbf{x})$ are all discretized and mapped into one-dimensional arrays using Eq. (15).

For the regularization operator $\mathbf{D}$, we consider the identity operator $\mathbf{I}$ and the finite-difference approximation to the Laplacian operator, which involves 6 s-order partial derivatives, $\partial^2/\partial x^2, \partial^2/\partial y^2, \partial^2/\partial z^2, \partial^2/\partial x\partial y, \partial^2/\partial x\partial z, \partial^2/\partial y\partial z$. These partial derivatives can be approximated using a central finite-difference scheme. Considering Eq. (15), the finite-difference approximation for each of the partial derivatives can be generated using a nested-loop (Fig. 4a) and the regularization matrix $\mathbf{A}_d$ can be obtained by concatenating the finite-difference approximations to each of the partial derivatives and the identity matrix (Fig. 4b). However, the regularization matrix generated using this straightforward approach has a bandwidth that is unnecessarily large. The bandwidth of $\mathbf{A}_d$ can be reduced significantly by combining the 7 nested-loops for generating the identity matrix and the finite-difference approximations to the 6 partial derivatives into one nested-loop (Fig. 4a). The resulting regularization matrix $\mathbf{A}_d$ is shown in Fig. 4c. The band itself is also highly sparse (Fig. 4d). The interval between the first and the last diagonal is around 2 $(n_x+n_x\times n_y)$.

The bandwidth of the regularization matrix $\mathbf{A}_d$ can be reduced even further by employing more sophisticated re-ordering algorithms (e.g., Cuthill and McKee, 1969; Gibbs et al., 1976; Rosen, 1968), which may involve permutations in both rows and columns. Since the values in vector $\mathbf{d}$ that correspond the damping matrix are all zeros, the row permutation on damping matrix does not cause any additional changes. However, applying column permutation to damping matrix requires correspondingly reordering to the kernel matrix and solution vector and therefore introduces significant overhead. Considering the overhead involved in such re-ordering algorithms, at the current stage we prefer the row permutation algorithm as shown in Fig. 4.

2.5. Load balancing

To improve the overall performance of our code, we need to balance the amount of data stored on each processor (i.e., memory balance) and the number of calculations performed by each processor (i.e., computation balance) while minimizing the amount of inter-processor communication overhead.

In practical seismic tomography applications, depending upon the spatial distributions of the seismic sources and the seismic receivers, the number of non-zero elements in the Jacobian matrix $\mathbf{A}_k$ can vary significantly from row to row and from column to column. In general, the number of non-zero elements is fewer for misfit measurements made at shorter source-receiver paths (i.e., rows in $\mathbf{A}_k$) and also fewer for regions (i.e., columns in $\mathbf{A}_k$) crossed by fewer source-receiver paths. A direct result of such a highly uneven distribution of non-zero elements in the Jacobian matrix $\mathbf{A}_k$ is that a straightforward even partition of $\mathbf{A}_k$ based on either the number of rows or the number of columns will result in a highly unbalanced memory utilization across all processors (Huang et al., 2013). For the full-wave tomography analyzed in this study, Fig. 5a shows the distribution of the number of non-zero elements of $\mathbf{A}_k$ on each processor based on an even partition of all the columns among 4000 processors. Since the matrix–vector multiplication operations $\mathbf{Av}$ and $\mathbf{A}^T\mathbf{u}$ on each processor are mainly determined by the number of non-zero elements of $\mathbf{A}_k$ on each processor, such a highly unbalanced distribution of non-zero elements also results in an unbalanced distribution of computational load among all processors.

A partition scheme that can produce a more balanced memory utilization pattern is based on the number of non-zero elements in $\mathbf{A}_k$ and $\mathbf{A}_d$. Suppose the total number of non-zero elements in $\mathbf{A}_k$ is N_k and the total number of non-zero elements in $\mathbf{A}_d$ is N_d, on each processor we would like to store around N_k/N_p non-zero elements of $\mathbf{A}_k$ and around N_d/N_p non-zero elements of $\mathbf{A}_d$. Since we are storing both $\mathbf{A}_d^p$ and $\mathbf{A'}_d^p$ on each processor, on average we need about $2N_d/N_p$ memory units for the regularization matrix on each processor. Each processor also stores an entire copy of $\mathbf{u}_k$, which has M_k elements, a portion of $\mathbf{u}_d$, which has M_d elements in total, and a portion of $\mathbf{v}$, which has N elements in total. The vectors $\mathbf{u}'$ and $\mathbf{v}'$ can share the same memory allocations with $\mathbf{u}$ and $\mathbf{v}$ respectively. To ensure a balanced utilization of the memory, the preferred number of non-zero elements for all matrices and vectors on each processor should be around

$$\overline{N}_e = (N_k+2N_d+M_d+N)/N_p+M_k. \tag{16}$$

We start our partition process by assigning columns of $\mathbf{A}_k$ to the first processor one by one. For the first processor, $c_b^1=1$. Since $\mathbf{A}_k$ is stored in the CSC format, the number of non-zero elements of $\mathbf{A}_k$ assigned to the first processor can be easily counted as we keep assigning more columns to the first processor. On the c_b^1-th column of the re-ordered $\mathbf{A}_d$ matrix there are a number of non-zero elements. We set r_b^1 to be the median of the row numbers of those non-zero elements (Fig. 3). As we increase the number of columns of $\mathbf{A}_k$ assigned to the first processor one by one, we also increase the number of rows of $\mathbf{A}_d$ assigned to the first processor one by one. When we assign the c-th column of $\mathbf{A}_k$ to the first processor, we also assign the r-th row of $\mathbf{A}_d$ to the first processor and r is chosen to be the median of the row numbers of the non-zero elements on the c-th column of $\mathbf{A}_d$. Since $\mathbf{A}_d$ is stored in the CSR format, the number of non-zero elements of $\mathbf{A}_d$ assigned to the first processor can also be easily counted. The elements in $\mathbf{u}_d$ and $\mathbf{v}$ are also assigned to the first processor one by one as we increase c and r. When the total number of non-zero elements assigned to the first processor N_e^1 exceeds $q_1\overline{N}_e$, where q_1 is a user-specified parameter, we stop assigning non-zero elements to the first processor and start to assign the rest of the matrices and vectors to the second processor. The column number c_e^1 is set to the current c value and the row number r_e^1 is set to the current r value (Fig. 3). For the second processor, $c_b^2=c_e^1+1$ and $r_b^2=r_e^1+1$. This process continues until all the non-zero elements in the matrices and vectors are assigned. For the full-wave tomography analyzed in this study, when $q_1=1$ we obtain a perfectly balanced memory utilization pattern (Fig. 5c). However, the number of columns of $\mathbf{A}_k$ and also the number of rows of $\mathbf{A}_d$ assigned to each processor can vary significantly from processor to processor (Fig. 5d).

If processor p has more columns of $\mathbf{A}_k$ than other processors, it is likely that processor p also owns more rows of $\mathbf{A'}_d$ and $\mathbf{A}_d$, since r_b^p and r_e^p are determined from c_b^p and c_e^p. As shown on Fig. 3, a direct consequence of owning more rows of $\mathbf{A'}_d$ and $\mathbf{A}_d$ is that processor p needs to gather more elements of the vectors $\mathbf{u}$ and $\mathbf{v}$ from other processors when computing the matrix–vector multiplications $\mathbf{A'}_d\mathbf{u}$ and $\mathbf{A}_d\mathbf{v}$. For the particular structure of the regularization matrix as shown in Fig. 2, processor p needs to gather more elements of vectors whose corresponding multiplicands are in the second and the third diagonals of the regularization matrix and those elements of $\mathbf{u}$ and $\mathbf{v}$ are usually located on many different processors. Since all processors need to be synchronized before initiating the calculation in step (2b) (Fig. 1), the inter-processor communication overhead caused by a few

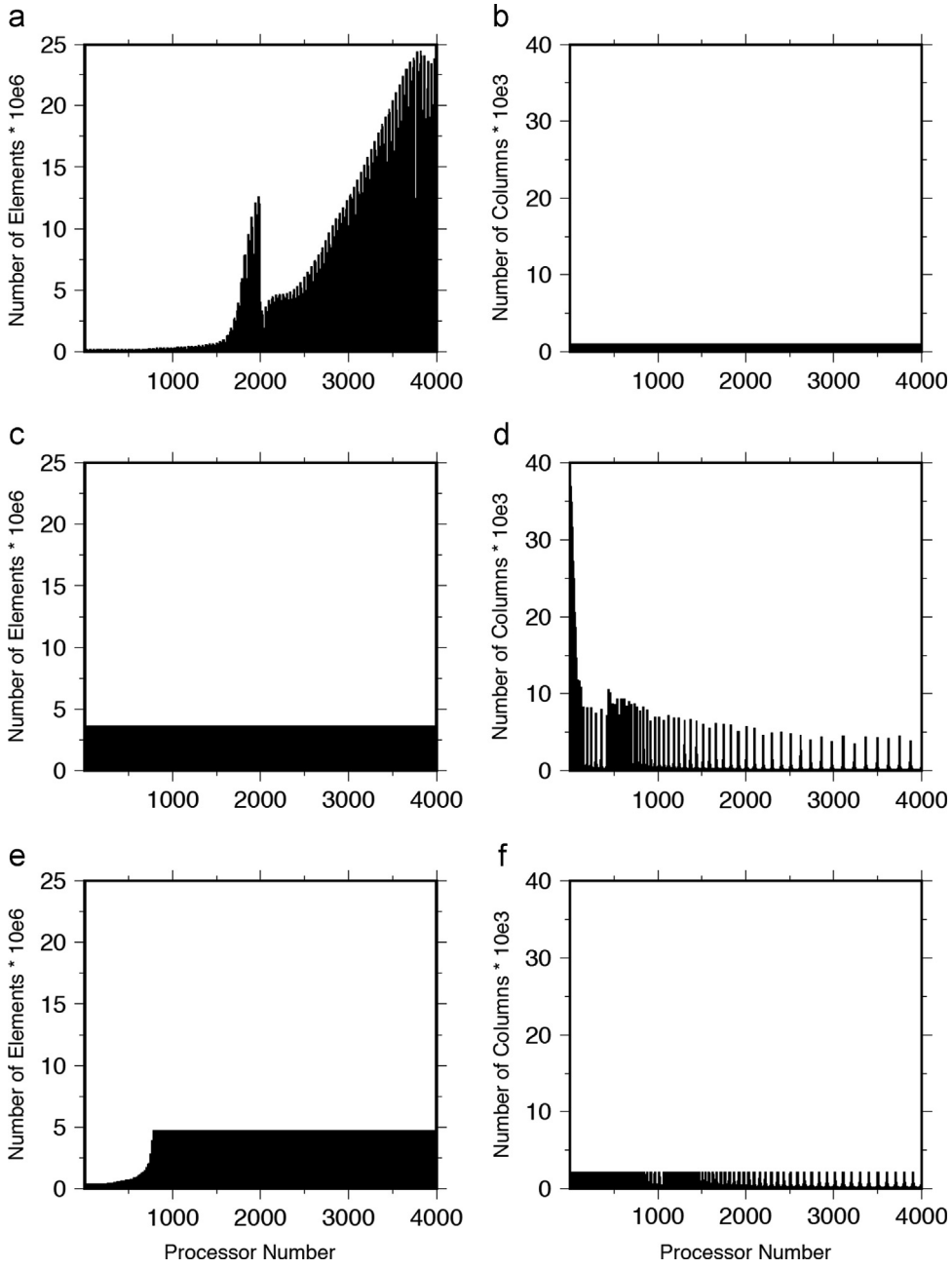


Fig. 5. Histograms of element and column loadings on each processor for the even column (a and b), load balancing (c and d) and optimal (e and f) partition methods. (a) max_elem/avg_elem=6.71, (b) max_col/avg_col=1.00, (c) max_elem/avg_elem=1.00 (d) max_col/avg_col=38.69, (e) max_elem/avg_elem=1.30 and (f) max_col/avg_col=2.18.

processors who have many more columns of $\mathbf{A}_k$ than other processors may delay the progress of the whole calculation. To avoid such a problem, when assigning the columns of $\mathbf{A}_k$ to each processor, we also require that the total number of columns on each processor N_c^p does not exceed $q_2 N/N_p$, where q_2 is a user-specified parameter. When distributing non-zero elements of the matrices and vectors to processor p, we stop the process if one of the following two conditions is satisfied,

$$N_e^p \geq q_1 \overline{N}_e, \tag{17}$$

$$N_c^p \geq q_2 N/N_p. \tag{18}$$

For the tomographic inversion analyzed in this study, we obtained the shortest overall wall-time when setting $q_1=1.30$ and $q_2=2.18$ on the Yellowstone supercomputer at NWSC. The distributions of N_e^p and N_c^p for all processors are shown in Fig. 5e and f.

Table 1
Characteristics of datasets.

	12K kernel dataset	125K kernel dataset
N	38,093,067	38,093,067
M_k	12,000	125,520
M_d	261,330,576	261,330,576
nz $\mathbf{A}_k$	14,190,626,015	143,524,414,175
nz $\mathbf{A}_d$	818,542,016	818,542,016

3. Results

We have examined the scalability of our code and made performance comparisons with LSQR solver of PETSc on Yellowstone supercomputer at the NCAR-Wyoming Supercomputing

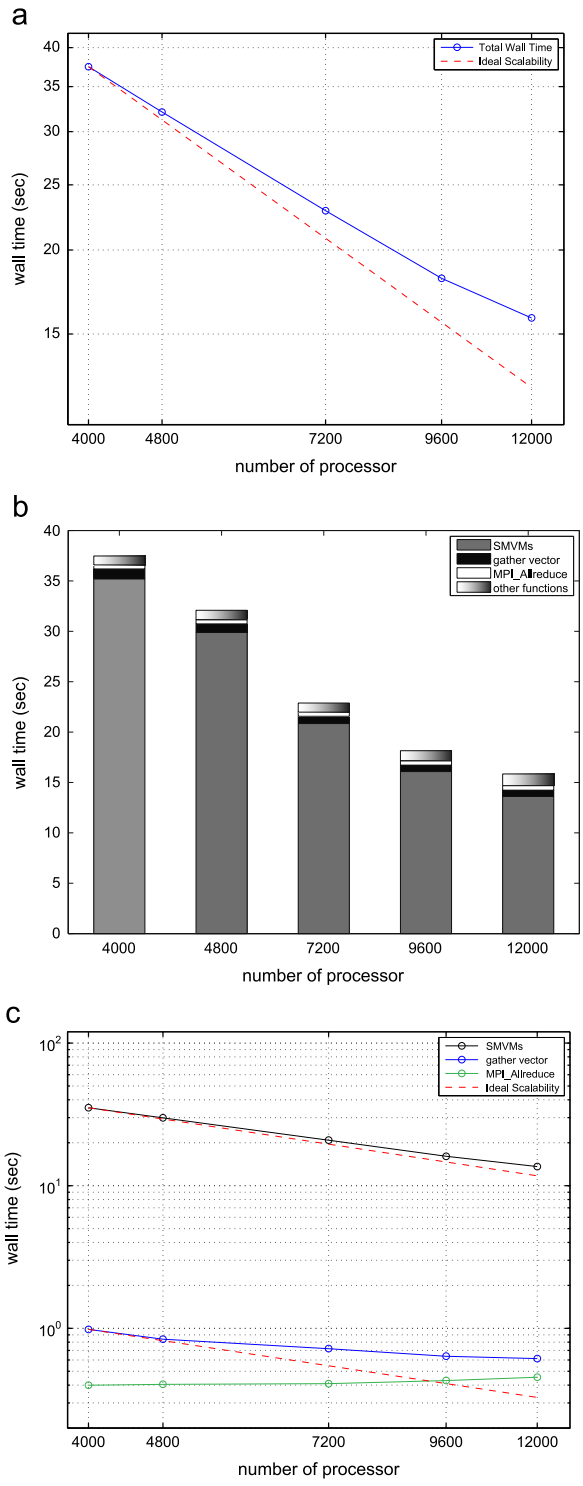


Fig. 6. Performance analysis results of 100 LSQR iterations of the 125K kernel dataset from 4000 to 12,000 processors. (a) The comparisons between our wall time measurements and ideally scaled wall time. (b) The stacked histograms show the wall time of different operations in our implementations. In all cases, over 90% of total wall time is spent on sparse matrix–vector multiplications (SMVMs). (c) The wall times of SMVMs and communication operations. The red dash lines represent the predicted time of ideal scalability. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

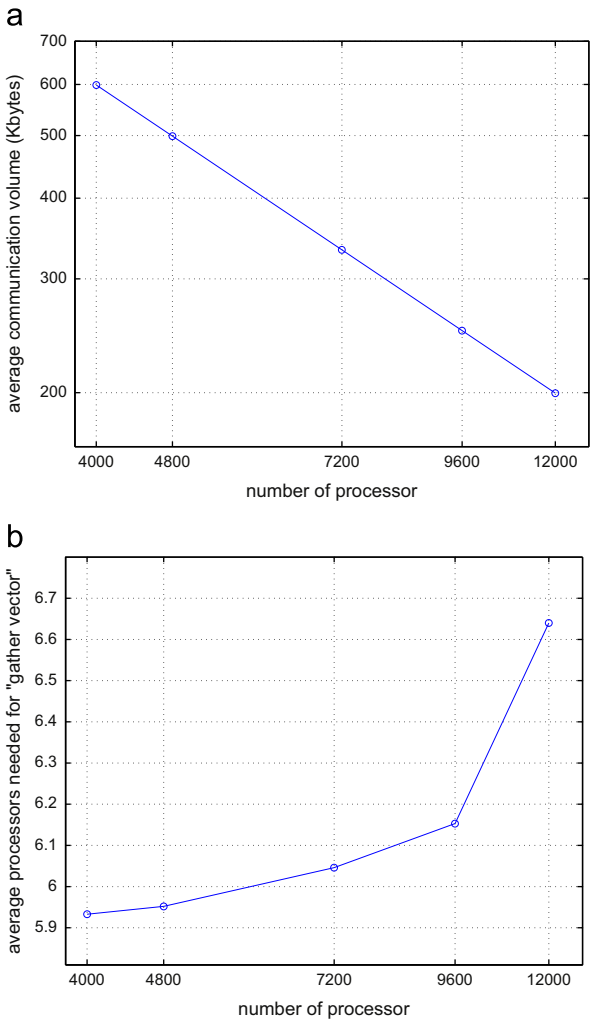


Fig. 7. (a) The average communication volume of gathering required vector elements from the other processors and (b) average processors that each processor need to communicate to during the $\mathbf{A}_d\mathbf{v}$ and $\mathbf{A}_d^T\mathbf{u}$ operations from 4000 to 12,000 processors.

Center (NWSC) in Cheyenne, Wyoming. The Yellowstone system is based on IBM's iDataPlex architecture with 4518 dual-socket nodes. There are 8 Intel 2.6-GHz Intel Sandy Bridge EP with Advanced Vector Extensions (AVX) processors per socket and 2 GB 1600-MHz DDR3 memory per processor. We used all 16 processors per node in all the following tests.

The details of two datasets of our full-3D waveform tomography for Southern California used in the tests are listed in Table 1. In the inversion problems, the zeroth-order and second-order Tikhonov regularization matrices are used (Section 2.4 and Fig. 4) and the size of matrix **A** is about 261 million rows by 38 million columns (Table 1). The 125K kernel dataset contains all 125,520 frequency-dependent kernels and there are over 143 billion non-zero elements in matrix **A** (Table 1). We will use the 125K kernel dataset for our performance analysis tests. In this paper, we compared the performances between our LSQR code and the parallel LSQR solver of PETSc (Balay et al., 1997). Due to the memory limitation on the PETSc, we picked about one tenth of kernels in 125K kernel dataset for the performance comparison tests. In the 12K kernel dataset, the size of inversion problem is about the same, but the amount of non-zero elements is about one tenth of the 125K kernel dataset (Table 1).

3.1. Performance analysis

We first evaluate the scalability of our implementations by using the 125K kernel dataset and all the measurements correspond to the wall time of 100 LSQR iterations. Fig. 6a shows the

comparisons between our wall time measurements (the blue curve) and the scalable predictions (the red curve) from 4000 to 12,000 processors. The differences between measured and predicted times ((measurement-prediction)/prediction) are about 2.8% on 4800 processors, 9.9% on 7200 processors, 16.3% on 9600 processors, and 26.8% on 12,000 processors. In general, the scalability is weaker when the number of processor increases. To understand the cause of this issue, a further time profiling of operations in our implementations is required.

We first analyze the time spending of the operations in our implementations. In Fig. 6b, the total wall times from 4000 to 12,000 processors are separated into four categories. The "SMVMs" represents the time spending on the pure sparse matrix–vector multiplications (SMVMs) during the $\mathbf{Av}$ and $\mathbf{A}^T\mathbf{u}$ and excludes any other expenses. The "gather vector" and "MPI_Allreduce" are the time spending on two types of inter-processor communication. The "gather vector" includes the wall times of gathering required vector $\mathbf{v}$ elements from the other processors for $\mathbf{A}_d\mathbf{v}$ and gathering required vector $\mathbf{u}$ elements from the other processors for $\mathbf{A}_d^T\mathbf{u}$ (details in Section 2.2). In our implementations, the other type of communication is summing the $\mathbf{u}'_k$ from all processors and then distributing the updated $\mathbf{u}'_k$ back to all processors during the matrix–vector multiplication $\mathbf{A}_k\mathbf{v}$ (details in Section 2.2). In practice, we used the "MPI_Allreduce" function, which is more efficient than a combined use of "MPI_reduce" and "MPI_Bcast", to complete the task. The wall times of the rest of expenses, such as vector normalizations and scalar-vector products are classified into the "other functions" category. In all the tests, over 95% of the total wall time is used in the matrix–vector multiplications in the LSQR iterations, which include the sparse matrix–vector multiplications (SMVMs) and communication ("gather vector" and "MPI_Allreduce") operations (Fig. 6b).

Since the majority of wall time is spent on the sparse matrix–vector multiplications and communication in our implementations, we further analyze the scalabilities of those operations. Fig. 6c shows the wall times of SMVMs (the black curve), "gather vector" (the blue curve) "MPI_Allreduce" (the green curve) and the scalable predictions (the red curves) from 4000 to 12,000 processors. For the SMVMs, the time differences between measurements and predictions ((measurement-prediction)/prediction) are about 1.9% on 4800 processors, 6.6% on 7200 processors, 9.6% on 9600 processors, and 16.1% on 12,000 processors (Fig. 6c). Although the scalability is weaker when the number of processors increases, the differences are all less than 20%. For the "gather vector", the time differences between measurements and predictions are about 2.4% on 4800 processors, 31.9% on 7200 processors, 55.4% on 9600 processors, and 87.2% on 12,000 processors (Fig. 6c). Compare to the SMVMs, the scalability of the "gather vector" operations is much poorer. Fig. 7a and b show the average communication volume during the $\mathbf{A}_d\mathbf{v}$ and $\mathbf{A}_d^T\mathbf{u}$ operations and the average number of processors needed during the "gather vector" operations for each processor. As processors increase, the average communication volume decreases about linearly (Fig. 7a) but the average number of processors needed during communication increases accordingly (Fig. 7b). The increase in the average number of processors needed during the "gather vector" operations (Fig. 7b) implies that the total processors involved in the communication operations also increase and therefore limits the scalability of the "gather vector" operations. For the "MPI_Allreduce" function, its wall time is not scalable. When the number of processors increases, there are more processors involved in the operations. However, the increase in wall time is small (Fig. 6c).

Although the scalability of our implementations is weaker as processors increase, the overall wall time still decreases (Fig. 6a and b). This is because the majority of total wall time (about 90% in all the tests) is spent on the SMVMs (Fig. 6b), which has relatively strong scalability (Fig. 6c).

3.2. Performance comparisons

The Portable, Extensible Toolkit for Scientific Computation (PETSc) (Balay et al., 1997) is a widely used computational library. In PETSc library, many easy-to-use linear equations solvers are provided and the LSQR algorithm is in the Krylov Subspace

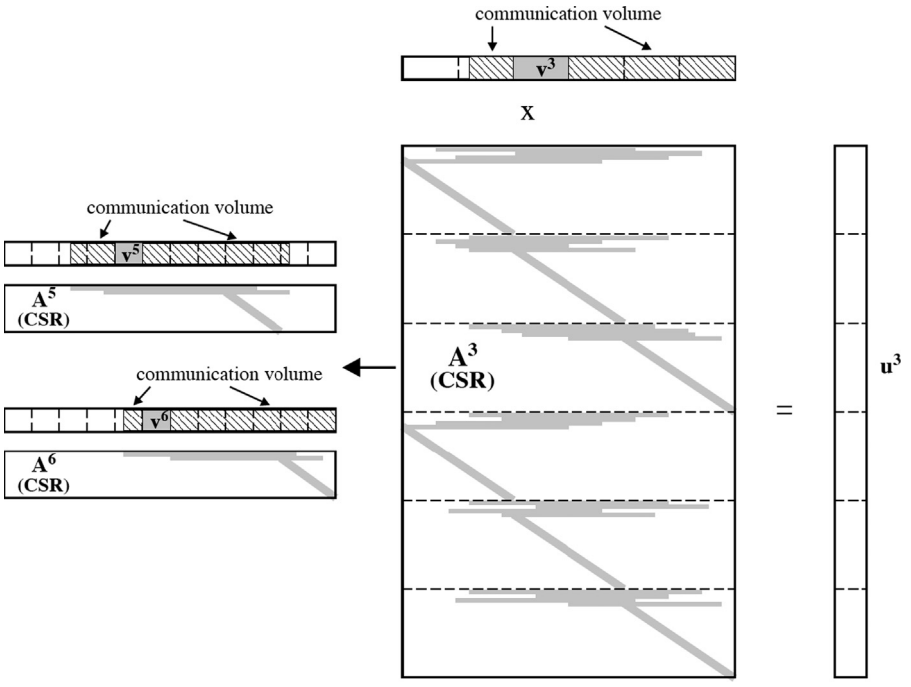


Fig. 8. Illustration of PETSc's matrix–vector multiplication implementations. The gray bars in matrix $\mathbf{A}$ represent the non-zero values of kernel and damping matrices. The required communication volume is in strip-pattern. For memory and computation balance, each processor stores a portion of kernel and damping matrices. When the number of processor double, the matrix–vector multiplication of $\mathbf{A}^3\mathbf{v}^3$ is paralleled into $\mathbf{A}^5\mathbf{v}^5$ and $\mathbf{A}^6\mathbf{v}^6$, as shown in left-hand-side.

Methods (KSP) component. To use the LSQR solver in PETSc library, we used functions in PETSc library to create the input matrix and vectors. The non-zero elements in the matrix **A**, including the Jacobian (or Fréchet) matrix $\mathbf{A}_k$ and the regularization matrix $\mathbf{A}_d$, are stored in the default matrix format, the compressed-sparse-row (CSR) format. Each processor stores both a portion of matrix $\mathbf{A}_k$ and matrix $\mathbf{A}_d$ (Fig. 8) for both memory and calculation balances. The vector **v** and **u** are evenly separated into processors (Fig. 8). Processor p owns sub-matrix $\mathbf{A}^p$, a portion of vector **v**, $\mathbf{v}^p$, and a portion of vector **u**, $\mathbf{u}^p$. During the matrix–vector multiplication **Av**, because each processor only owns a portion of vector **v**, it is required to transfer needed elements of vector **v** from other processors to processor p to complete the matrix–vector multiplication $\mathbf{A}^p\mathbf{v}$ (Fig. 8). The communication volume for processor p depends upon the number of required **v** elements for $\mathbf{A}^p\mathbf{v}$ outside the $\mathbf{v}^p$(Fig. 8). When the required elements outside the $\mathbf{v}^p$ is large, the increases not only in the communication cost but also in the memory allocation for storing the gathered vector elements from other processors. The same amount of communication cost is required during the matrix–vector multiplication $\mathbf{A}^T\mathbf{u}$.

In this paper, we compared the performances between our and the PETSc's implementations of the LSQR algorithm. Due to the memory constraints on PETSc's implementations, the 12 K kernel dataset is used for comparison tests. We used a built-in profiling option in the PETSc that can be accessed by adding the flag "-log_summary". Fig. 9a shows total and communication wall times of two implementations for 100 LSQR iterations from 2400 to 9600 processors. In PETSc's implementations, the wall times increase when the number of processor increases and over 50% of total wall time is spent on communication. We then compare the average communication volume between two implementations (Fig. 9b). Notice that we include the communication volume from both "gather vector" and "MPI_Allreduce" operations (Section 2.2) in the comparison tests. When the number of processor increases, the average communication volume decreases in our implementations but that increases in PETSc's implementations.

In PETSc's implementations, when the number of processor increases, the wall time of communication and average communication volume per processor also increase (Fig. 9). Fig. 8 illustrates the increase of communication volume, when the number of processor is doubled. To balance the memory loading and calculations, the original task in each processor is paralleled into two processors. In the example, the original matrix–vector multiplication $\mathbf{A}^3\mathbf{v}$ in the third processor is partitioned into two sub-matrix–vector multiplications $\mathbf{A}^5\mathbf{v}$ and $\mathbf{A}^6\mathbf{v}$. It turns out that the amount of communication volume for $A^5\mathbf{v}$ and $\mathbf{A}^6\mathbf{v}$ is more than that of the $\mathbf{A}^3\mathbf{v}$. In addition, each processor owns a smaller portion of vector **v**, so the number of processor involved in communication also increases. Due to the increase in both communication volume and processor number, the total communication wall time also increases. Compare to PETSc's implementations, even though the scalability of

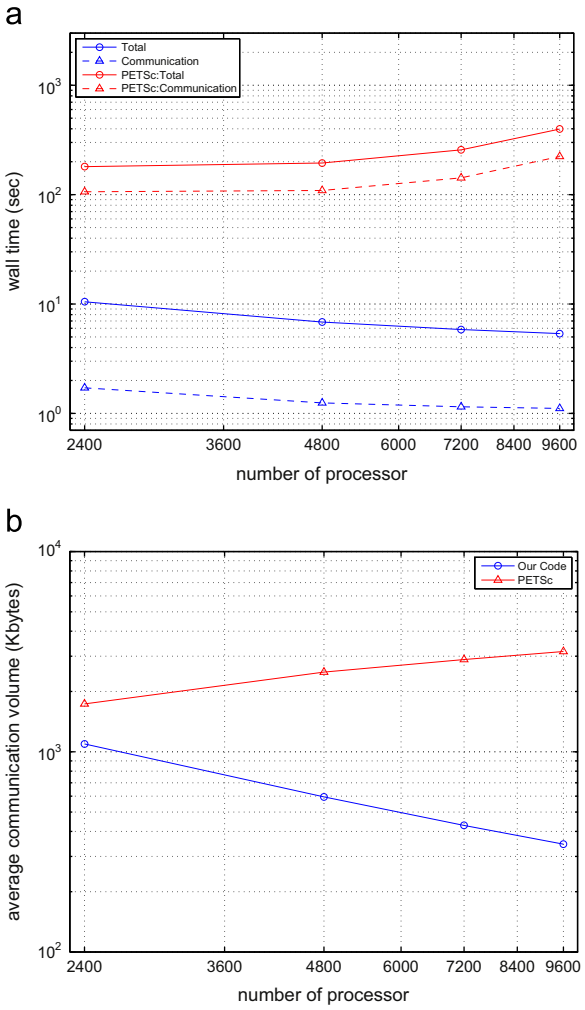


Fig. 9. Comparisons between our and PETSc's implementations of the LSQR algorithm for 100 LSQR iterations of the 12K kernel dataset from 2400 to 9600 processors. (a) Total and communication wall time and (b) the average communication volume of two implementations.

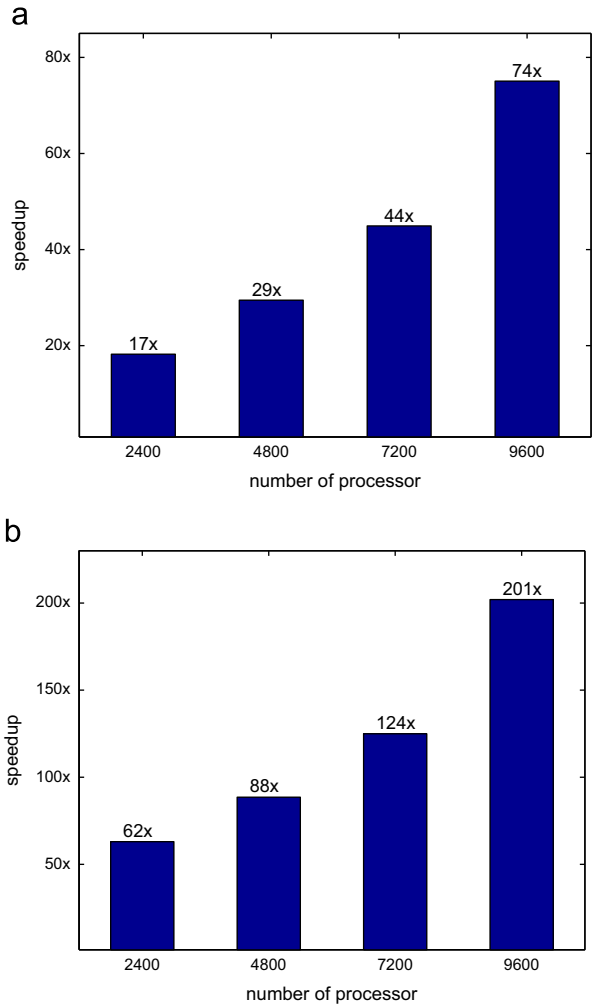


Fig. 10. Histograms show the speedups of our implementation in (a) total wall time and (b) communication wall time.

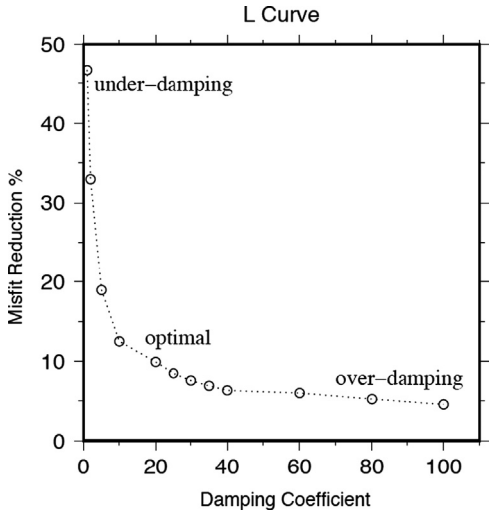


Fig. 11. The L-curve of the full-3D waveform tomographic inversion for Southern California.

communication wall time is weak in our implementations, when the number of processor increases, the communication wall time still decreases (Fig. 9a).

In Fig. 10 we show the speedups (wall time of PETSc/wall time of our code) of the total and communication wall times from 2400 to 9600 processors. The speedups of total wall time are about 17x on 2400 processors, 29x on 4800 processors, 44x on 7200 processors and 74x on 9600 processors (Fig. 10a). The majority of improvements are from the communication processes and the Fig. 10b shows the speedups of communication wall time. The speedups of communication wall time increase from 62x on 2400 processors to 201x on 9600 processors (Fig. 10b).

3.3. *Full-3D waveform tomography for Southern California*

To find an optimal damping coefficient, many LSQR runs with different damping coefficients are required in real seismic tomographic inversions. Our optimized codes have significantly reduced the required time and computational resources for the LSQR algorithm and therefore make this process feasible for very large

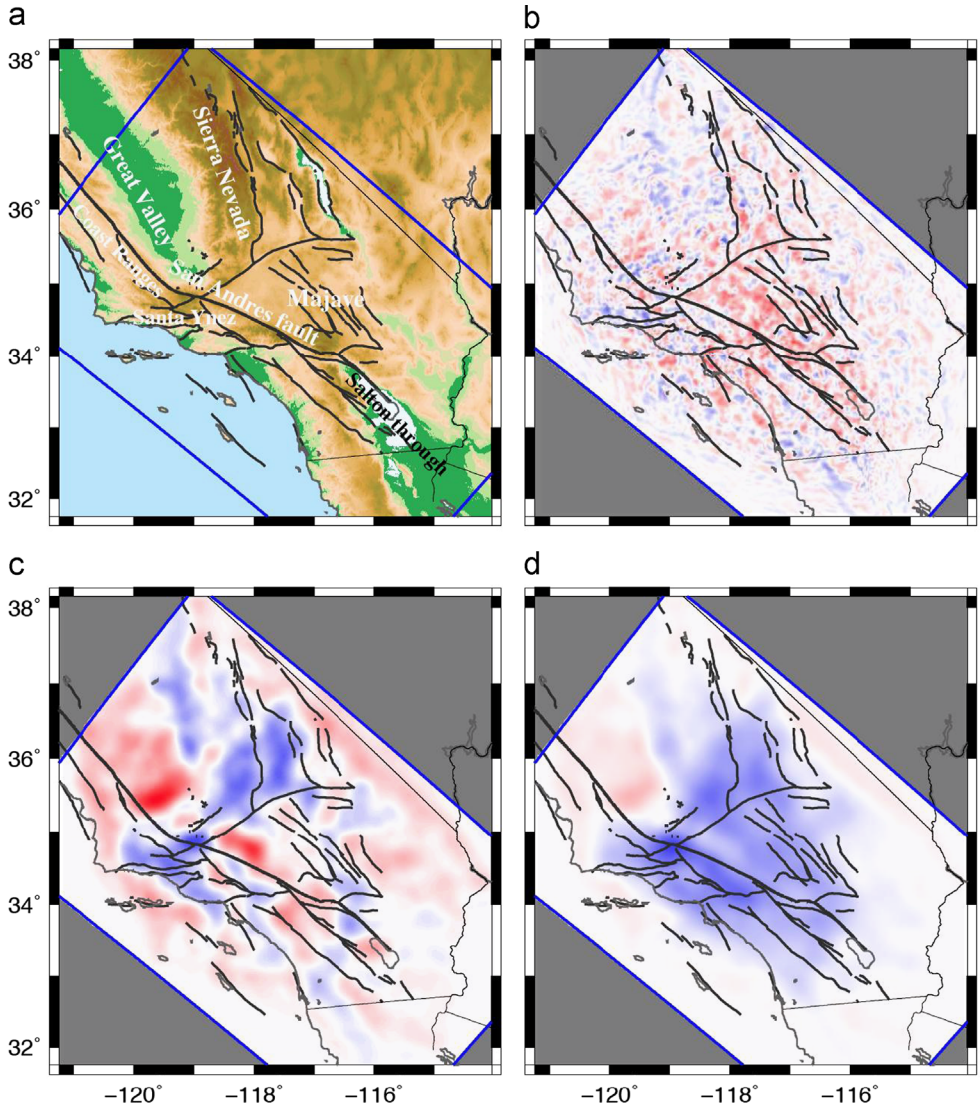


Fig. 12. (a) The map shows the topography and major faults (thick black lines) of Southern California. (b–d) The under-damping, optimal and over-damping LSQR perturbation results at 0.5 km of the Southern California tomographic inversion. (a) Topography, (b) under-damping, (c) optimal, (d) over-damping. In perturbation maps, the red regions represent velocity reduction areas and the blue regions represent velocity increase areas. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

seismic tomographic inversions. The result curve is usually characterized by its L shape and is called an "L-curve" (e.g. Aster et al., 2005). In general, a damping coefficient close to the corner of the L-curve is selected as an optimal value. Fig. 11 shows the L-curve of our Southern California tomographic inversion.

Fig. 12a shows the topography and major faults (thick black lines) in our study area and Fig. 12b–d show the perturbation results of different damping coefficients at 0.5 km. In the under-damping result (Fig. 12b), although the misfit reduction is larger, the perturbations are oscillated. In the optimal result (Fig. 12c), the perturbations show many correlations with geological structures. For example, the seismic velocity in the southern Great Valley, offshore basins, Salton trough and Majave regions where east of the San Andreas fault need to reduce, but the seismic velocity in the Sierra Nevada and Coast Ranges where close to the Santa Ynez regions need to increase. In the over-damping result, not only the misfit reduction is small but also the perturbations are too smooth (Fig. 12d).

4. Conclusions and future developments

The LSQR algorithm is efficient and stable for solving large and ill-posed linear systems and widely used in seismic tomographic inversions. Since the increase in seismic observations and advances in computational seismology have made seismic tomographic inversions much larger than they are before (e.g. Lin et al., 2010; Chen et al., 2007b), an efficient and paralleled LSQR solver is required for those seismic tomographic inversions. In this paper, we present our optimizations on our LSQR code and discuss the benefits of our optimizations. The use of re-ordered damping matrix simplifies the communication and reduces the amount of communication volume among processors. The combination of using CSC and CSR formats and the MPI I/O has made the data reading process extremely efficient for very large datasets. To further improve the performance of the LSQR code, we utilize an optimal partition method to balance the amount of data loading, calculations and communication volume among the processors.

In this paper, the performance of our and PETSc's implementations of the LSQR algorithm was compared. We use the kernel dataset of full-3D waveform tomographic inversion for Southern California in the tests. The size of the inversion problem is about 261 million rows by 38 million columns and the speedups of total wall time for 100 LSQR iterations varies from 17x to 74x. Because our implementations of the LSQR algorithm are designed for reducing the communication cost in a large inverse problem, the speedup of communication wall time also varies from 62x to 201x. In addition, our implementations of LSQR algorithm require less memory usage when compare with PETSc's implementations.

In future work, we could like include the OpenMP in our currently implementations. In addition, we also plan to use the Graphics Processing Units (GPUs) in future developments. We would like to investigate combining OpenCL or CUDA with our MPI code to improve the performance further.

Acknowledgment

The kernel calculations in this research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U. S. Department of Energy under contract DE-AC02-06CH11357. The LSQR tests used resources of the Yellowstone supercomputer at the NCAR-Wyoming Supercomputing Center (NWSC). The work was supported in part by NSF under CAREER-1054834 and 0941735.

References

Aki, K., Richards, P.G., 2002. Quantitative Seismology, 2nd ed. University Science Books, Sausalito, California.
Aster, R.C., Borchers, B., Thurber, C.H., 2005. Parameter Estimation and Inverse Problems. Elsevier Academic Press, Amsterdam.
Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H. (Eds.), 2000. SIAM, Philadelphia.
Backus, G.E., Gilbert, F., 1968. The resolving power of gross earth data. Geophysical Journal of the Royal Astronomical Society 16, 169–205.
Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F., 1997. Efficient management of parallelism in object-oriented numerical software libraries, Modern Software Tools for Scientific Computing. Birkhäuser, Boston, MA, pp. 163–202.
Bamberger, A., Chavent, G., Hemon, C., Lailly, P., 1982. Inversion of normal incidence seismograms. Geophysics 47, 757–770.
Bamberger, A., Chavent, G., Lailly, P., 1977. Une application de la théorie du contrôle à un problème inverse sismique. Annales Geophysicae 33, 183–200.
Bao, H., Bielak, J., Ghattas, O., Kallivokas, L.O.D., 1998. Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers. Computer Methods in Applied Mechanics and Engineering 152, 85–102.
Červený, V., 2005. Seismic Ray Theory. Cambridge University Press, Cambridge.
Chen, P., Jordan, T.H., Zhao, L., 2007a. Full three-dimensional tomography: a comparison between the scattering-integral and adjoint-wavefield methods. Geophysical Journal International 170, 175–181.
Chen, P., Zhao, L., Jordan, T.H., 2007b. Full 3D tomography for the crustal structure of the Los Angeles region. Bulletin of the Seismological Society of America 97, 1094–1120.
Cuthill, E., McKee, J., 1969. Reducing the bandwidth of sparse symmetric matrices. Presented at the 1969 24th National Conference, ACM Press, New York, New York, USA, pp. 157–172.
Dahlen, F., Hung, S., Nolet, G., 2000. Frechet kernels for finite-frequency travel-times-I. Theory. Geophysical Journal International 141, 157–174.
Dumbser, M., Kaser, M., Toro, E.F., 2007. An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes–V. Local time stepping and p-adaptivity. Geophysical Journal International 171, 695–717.
Gibbs, N.E., Poole Jr, W.G., Stockmeyer, P.K., 1976. An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM Journal on Numerical Analysis 13, 236–250.
Graves, R.W., 1996. Simulating seismic wave propagation in 3D elastic media using staggered-grid finite differences. Bulletin of the Seismological Society of America 86, 1091–1106.
Huang, H., Dennis, J.M., Wang, L., Chen, P., 2013. A scalable parallel LSQR algorithm for solving large-scale linear system for tomographic problems: a case study in seismic tomography. In: Proceedings of the 2013 International Conference on Computational Science (ICCS). Procedia Computer Science.
Huang, H., Wang, L., Lee, E.J., Chen, P., 2012. An MPI-CUDA implementation and optimization for parallel sparse equations and Least Squares (LSQR). In: Procedings of the 2012 International Conference on Computational Science (ICCS). Procedia Computer Science.
Hung, S., Dahlen, F., Nolet, G., 2000. Frechet kernels for finite-frequency traveltimes-II. Examples. Geophysical Journal International 141, 175–203.
Komatitsch, D., Liu, Q., Tromp, J., Suss, P., Stidham, C., Shaw, J., 2004. Simulations of ground motion in the Los Angeles basin based upon the spectral-element method. Bulletin of the Seismological Society of America 94, 187–206.
Komatitsch, D., Vilotte, J.-P., 1998. The spectral element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures. Bulletin of the Seismological Society of America 88, 368–392.
Lin, G., Thurber, C.H., Zhang, H., Hauksson, E., Shearer, P.M., Waldhauser, F., Brocher, T.M., Hardebeck, J., 2010. A California statewide three-dimensional seismic velocity model from both absolute and differential times. Bulletin of the Seismological Society of America 100, 225–240.
Liu, J., Liu, F., Liu, J., Hao, T., 2006. Parallel LSQR algorithms used in seismic tomography. Chinese Journal of Geophysics 49, 483–488.
Marquering, H., Dahlen, F.A., Nolet, G., 1999. Three-dimensional sensitivity kernels for finite-frequency traveltimes: the banana-doughnut paradox. Geophysical Journal International 137, 805–815.
Nolet, G., 1985. Solving or resolving inadequate and noisy tomographic systems. Journal of Computational Physics 61, 463–482.
Nolet, G., 1993. Solving large linearized tomographic problems, Seismic Tomography: Theory and Practice. Chapman & Hall, London, pp. 227–247.
Olsen, K., 1994. Simulation of Three-Dimensional Wave Propagation in the Salt Lake Basin. University of Utah, Salt Lake City, Utah.
Paige, C., Saunders, M., 1982. LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software (TOMS) 8, 43–71.
Pratt, R., Shin, C., Hicks, G., 1998. Gauss–Newton and full Newton methods in frequency-space seismic waveform inversion. Geophysical Journal International 133, 341–362.
Pratt, R.G., Worthington, M.H., 1990. Inverse theory applied to multi-source cross-hole tomography. Part 1: acoustic wave-equation method. Geophysical Prospecting 38, 287–310.
Rosen, R., 1968. Matrix bandwidth minimization. Presented at the 1968 23rd ACM National Conference, ACM Press, New York, New York, USA, pp. 585–595.
Tarantola, A., 1984. Inversion of seismic reflection data in the acoustic approximation. Geophysics 49, 1259–1266.

Tarantola, A., 1988. Theoretical background for the inversion of seismic waveforms including elasticity and attenuation. Pure and Applied Geophysics 128, 365–399.
Tarantola, A., 2005. Inverse Problem Theory and Methods for Model Parameter Estimation. Society for Industrial and Applied Mathematics, Philadelphia, USA.
Tromp, J., Tape, C., Liu, Q., 2004. Seismic tomography, adjoint methods, time reversal and banana-doughnut kernels. Geophysical Journal International 160, 195–216.
Zhao, L., Jordan, T., Chapman, C., 2000. Three-dimensional Fréchet differential kernels for seismic delay times. Geophysical Journal International 141, 558–576.
Zhao, L., Jordan, T., Olsen, K., Chen, P., 2005. Fréchet kernels for imaging regional earth structure based on three-dimensional reference models. Bulletin of the Seismological Society of America 95, 2066–2080.