# Parallel Minimum Norm Solution of Sparse Block Diagonal Column Overlapped Underdetermined Systems



F. SUKRU TORUN, Bilkent University
MURAT MANGUOGLU, Middle East Technical University
CEVDET AYKANAT, Bilkent University

Underdetermined systems of equations in which the minimum norm solution needs to be computed arise in many applications, such as geophysics, signal processing, and biomedical engineering. In this article, we introduce a new parallel algorithm for obtaining the minimum 2-norm solution of an underdetermined system of equations. The proposed algorithm is based on the Balance scheme, which was originally developed for the parallel solution of banded linear systems. The proposed scheme assumes a generalized banded form where the coefficient matrix has column overlapped block structure in which the blocks could be dense or sparse. In this article, we implement the more general sparse case. The blocks can be handled independently by any existing sequential or parallel QR factorization library. A smaller reduced system is formed and solved before obtaining the minimum norm solution of the original system in parallel. We experimentally compare and confirm the error bound of the proposed method against the QR factorization based techniques by using true single-precision arithmetic. We implement the proposed algorithm by using the message passing paradigm. We demonstrate numerical effectiveness as well as parallel scalability of the proposed algorithm on both shared and distributed memory architectures for solving various types of problems.

CCS Concepts: • **Mathematics of computing → Solvers**; **Mathematical software performance**; **Computations on matrices**; • **Computing methodologies → Shared memory algorithms**; **Massively parallel algorithms;**

Additional Key Words and Phrases: Minimum norm solution, underdetermined least square problems, parallel algorithms, balance method

**ACM Reference Format:**
F. Sukru Torun, Murat Manguoglu, and Cevdet Aykanat. 2017. Parallel minimum norm solution of sparse block diagonal column overlapped underdetermined systems. ACM Trans. Math. Softw. 43, 4, Article 31 (January 2017), 21 pages.
DOI: http://dx.doi.org/10.1145/3004280

## 1. INTRODUCTION

Underdetermined systems of equations [Lawson and Hanson 1987; Björck 1996] in which the minimum norm solution needs to be computed arise in many applications areas, such as geophysics [Zhdanov 2002; Sen and Stoffa 2013], signal and image processing [Bruckstein et al. 2009; Cotter et al. 2005], and biomedical engineering [Matsuura

Authors acknowledge PRACE, who provided Preparatory Access Call Type B (resource) awards for application numbered 2010PA2754. F. S. Torun was supported through the Scientific and Technological Research Council of Turkey (TUBITAK), under the program BIDEB-2211. M. Manguoglu was supported by Turkish Academy of Sciences Distinguished Young Scientist Award M.M/TUBA-GEBIP/2012-19.
Authors' addresses: F. S. Torun and C. Aykanat, Computer Engineering Department, Bilkent University, 06800, Ankara, Turkey; emails: sukruf@cs.bilkent.edu.tr, aykanat@cs.bilkent.edu.tr; M. Manguoglu, Computer Engineering Department, Middle East Technical University, 06800, Ankara, Turkey; email: manguoglu@ceng.metu.edu.tr.

DOI: http://dx.doi.org/10.1145/3004280

and Okabe 1995; Wang et al. 1992]. An underdetermined system of equations $Ax = f$ where $A$ is an $m \times n$ matrix with $m < n$ and $rank(A) = m$ has infinitely many solutions. In this article, we will focus on the minimum 2-norm solution of underdetermined linear least squares problems. This solution can be obtained either by direct or iterative methods. Examples of direct solvers typically compute the QR factorization of the transpose of the coefficient matrix $A$,

$$A^T = Q\begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{1}$$

Then, the minimum norm solution can be computed as,

$$x = Q\begin{pmatrix} R^{-T} f \\ 0 \end{pmatrix}, \tag{2}$$

which we will refer to as the "Q method," or without using $Q$,

$$x = A^T (R^T R)^{-1} f. \tag{3}$$

The latter approach is called the *seminormal equations (SNE)* [Gill and Murray 1973; Saunders 1972].

A considerable amount of effort has been spent on developing efficient parallel and sequential implementations of general sparse QR algorithms such as SuiteSparseQR [Davis 2011, 2013], HSL MA49 [Amestoy et al. 1996], SPOOLES [Ashcraft and Grimes 1999], and qr_mumps [Buttari 2013].

Other factorizations, such as LQ or SVD, can also be used for obtaining the minimum 2-norm solution of an underdetermined linear least squares problems. Another approach is to use the normal equations to obtain the minimum norm solution,

$$x = A^T (AA^T)^{-1} f, \tag{4}$$

which requires solution of a linear system. The solution of the linear system can be obtained directly via the Cholesky factorization or iteratively using a Krylov subspace method or any other iterative technique. Although normal and seminormal equation approaches could save some storage and computational costs, they have the potential of introducing numerical difficulties that can be disastrous in some cases when the problem is ill conditioned.

The Balance Scheme [Golub et al. 2001; Sameh and Sarin 2002; Tezduyar and Sameh 2006] is a parallel algorithm that was designed to solve an ill-conditioned, banded, linear system of equations that are sparse within the band. The rows of the linear system are partitioned into k blocks, where k is the number of processes. This partitioning gives rise to k linear least squares equations where each has infinitely many solutions. Since the coefficient matrix is banded, however, each block row has some columns that overlap with the neighboring blocks. The overlapping between the blocks means that parts of the solutions of the linear least squares problems cannot be independent, and the unique solution is obtained enforcing a constraint on the equality of the solution in the overlapping parts of the solution vector, giving rise to a smaller independent reduced system of equations, which is solved either directly or iteratively.

In this article, we show that the Balance Scheme can be extended to obtain the minimum 2-norm solution of an underdetermined system of equations. The algorithm designed for underdetermined systems in which the coefficient matrix is in a generalized banded form with column overlapping block diagonals that are sparse within the block.

There are a number of applications that give rise to coefficient matrices that are in column overlapping block diagonal or banded forms, such as Spline Interpolation

[Peyrovian and Sawchuk 1978], Seismic Tomography [Huang et al. 2013], and Linear Programming [Ho and Loute 1981; Fourer 1984]. We show that the proposed algorithm finds the minimum 2-norm solution of the linear least squares problems. Furthermore, Demmel and Higham [1993] have shown error bounds of the Q method and SNE for finding the minimum norm solution by simulating single-precision arithmetic in MATLAB [MATLAB 2015]. We confirm these bounds by using true single-precision arithmetic in MATLAB and show that the proposed algorithm is comparable to the Q method in MATLAB. Finally, since there are no sparse parallel QR factorization routines available for banded or a more generalized column overlapping block row matrices either on distributed or shared memory architectures, we compare the parallel scalability of the proposed algorithm against a well-known general parallel sparse QR factorization algorithm on two different parallel computing platforms. Since we use the same sparse QR factorization algorithm for the block diagonals, the proposed scheme can also be considered to be an extension of the general sparse QR factorization to distributed memory computing platforms for obtaining the minimum 2-norm solution of underdetermined linear least squares problems.

The rest of the article is organized as follows. Section 2 presents the formulation of proposed algorithm and its parallelization. In Section 3, we compare the proposed algorithm against a state-of-the-art algorithm in terms of parallel scalability and numerical accuracy on shared and distributed memory architectures. Finally, we confirm the error bounds for the underdetermined systems given in Demmel and Higham [1993] using true single precision and show the benchmarks of the Q method, SNE, and the proposed algorithm in Section 4.

## 2. THE PROPOSED ALGORITHM

In the first subsection, we first define the problem clearly and then explain the theory behind the proposed algorithm. In the second subsection, we discuss the details of the parallel algorithm and the implementation based on the theoretical findings given in the first subsection.

### 2.1. Formulation

An underdetermined system of equations of the form

$$Ax = f \tag{5}$$

is given, where $A$ is a sparse $m \times n$ $(m < n)$ matrix. We assume $A$ has full row rank. Our objective is to find the unique solution $x$ such that $||x||_2$ is minimized.

We assume that the coefficient matrix $A$ can be partitioned row-wise into $k$ diagonal blocks with column overlap as follows:

$$\begin{pmatrix} A_1 & B_1 & & & & & & & \\ & C_2 & A_2 & B_2 & & & & & \\ & & & C_3 & A_3 & B_3 & & & \\ & & & & & \ddots & \ddots & \ddots & \\ & & & & & & C_{k-1} & A_{k-1} & B_{k-1} \\ & & & & & & & C_k & A_k \end{pmatrix} \begin{pmatrix} x_1 \\ \xi_1 \\ x_2 \\ \xi_2 \\ \vdots \\ \vdots \\ x_{k-1} \\ \xi_{k-1} \\ x_k \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{k-1} \\ f_k \end{pmatrix}. \tag{6}$$

In Equation (6), diagonal blocks are defined as

$$\begin{aligned} E_1 &= (A_1, B_1) \\ E_i &= (C_i, A_i, B_i), \text{for } i = 2, \ldots, k-1 \\ E_k &= (C_k, A_k). \end{aligned} \tag{7}$$

Here, $B_i$ and $C_{i+1}$ denote the column overlapping submatrices of the successive $i^{th}$ and $(i+1)^{th}$ diagonal blocks. Let $t_i$ denote the size of the column overlap between the $i^{th}$ and $(i+1)^{th}$ diagonal blocks for $i = 1, \ldots, k-1$. In Equation (7), $A_i \in \mathbb{R}^{m_i \times n_i}$ for $i = 1, \ldots, k$, $B_i \in \mathbb{R}^{m_i \times t_i}$ for $i = 1, \ldots, k-1$, and $C_i \in \mathbb{R}^{m_i \times t_{i-1}}$ for $i = 2, \ldots, k$. Since $A \in \mathbb{R}^{m \times n}$, we have

$$\begin{aligned} m &= \sum_{i=1}^{k} m_i, \\ n &= \sum_{i=1}^{k} n_i + t, \end{aligned} \tag{8}$$

where $t = \sum_{i=1}^{k-1} t_i$ denotes the total column overlap size. Note that $E_1 \in \mathbb{R}^{m_1 \times \tilde{n}_1}$, $E_i \in \mathbb{R}^{m_i \times \tilde{n}_i}$ for $i = 2, \ldots, k-1$, and $E_k \in \mathbb{R}^{m_k \times \tilde{n}_k}$, where $\tilde{n}_1 = (n_1 + t_1)$, $\tilde{n}_i = (t_{i-1} + n_i + t_i)$ for $i = 2, \ldots, k-1$, and $\tilde{n}_k = (t_{k-1} + n_k)$.

As shown in Equation (6), the right-hand-side vector $f$ is partitioned comformably with the row block partition of the coefficient matrix. Hence, $f_i$ is a subvector of size $m_i$ for $i = 1, \ldots, k$. As also shown in Equation (6), the solution vector $x$ is partitioned comformably with the column block partition induced by the block diagonal form of the coefficient matrix. Thus, $x_i$ is a subvector of size $n_i$ for $i = 1, \ldots, k$, and $\xi_i$ is a subvector of size $t_i$ for $i = 1, \ldots, k-1$.

Underdetermined linear least squares problem (6) gives rise to $k$ smaller underdetermined linear least squares problems of the form

$$E_i z_i = f_i, \quad \text{for } i = 1, 2, \ldots, k, \tag{9}$$

where

$$\begin{aligned} z_1 &= (x_1^T, \xi_1^T)^T \\ z_i &= (\hat{\xi}_{i-1}^T, x_i^T, \xi_i^T)^T, \quad \text{for } i = 1, 2, \ldots, k-1 \\ z_k &= (\hat{\xi}_{k-1}^T, x_k^T)^T. \end{aligned} \tag{10}$$

The general solution of the system is

$$z_i = p_i + Q_i y_i, \quad \text{for } i = 1, 2, \ldots, k, \tag{11}$$

where $p_i$ is a particular solution that can be computed independently, $Q_i$ is a basis for $\mathcal{N}(E_i)$, and $y_i$ is arbitrary. One can obtain $Q_i$ via the $QR$ factorization

$$E_i^T = (\hat{Q}_i, Q_i) \begin{pmatrix} R_i \\ 0 \end{pmatrix}, \tag{12}$$

where $\hat{Q}_i$ and $Q_i$ have dimensions of $\tilde{n}_i \times m_i$ and $\tilde{n}_i \times (\tilde{n}_i - m_i)$ for $i = 1, \ldots, k$, respectively.

One way of obtaining $p_i$ is via the QR factorization just computed,

$$p_i = \hat{Q}_i ({R_i}^{-T} f_i). \tag{13}$$

Note that for $x$ to be one of the solutions of the original system, $\hat{\xi}_i$ must be equal to $\xi_i$ for $i = 1, \ldots, k-1$, which implies

$$\begin{aligned} p_1^{(low)} + (0, I)Q_1 y_1 &= p_2^{(up)} + (I, 0)Q_2 y_2 \\ p_2^{(low)} + (0, I)Q_2 y_2 &= p_3^{(up)} + (I, 0)Q_3 y_3 \\ \ldots &= \ldots \\ p_{(k-2)}^{(low)} + (0, I)Q_{k-2} y_{k-2} &= p_{(k-1)}^{(up)} + (I, 0)Q_{k-1} y_{k-1} \\ p_{(k-1)}^{(low)} + (0, I)Q_{k-1} y_{k-1} &= p_k^{(up)} + (I, 0)Q_k y_k, \end{aligned} \tag{14}$$

where $p_i^{(up)}$, $p_i^{(low)}$ and $p_i^{(rem)}$ refer to the upper, lower, and the remaining parts of the vectors $p_i$, respectively, as shown in the following,

$$p_1 = \begin{pmatrix} p_1^{(rem)} \\ p_1^{(low)} \end{pmatrix}, p_i = \begin{pmatrix} p_i^{(up)} \\ p_i^{(rem)} \\ p_i^{(low)} \end{pmatrix}, p_k = \begin{pmatrix} p_k^{(up)} \\ p_k^{(rem)} \end{pmatrix}. \tag{15}$$

Note that vectors $p_i^{(low)}$ and $p_{i+1}^{(up)}$ both have size $t_i$. $(I, 0)$ and $(0, I)$ refer to matrices of size $t_i \times (\tilde{n}_i - m_i)$ where $I$ is the identity matrix of size $t_i \times t_i$.

Equation (14) gives rise to a reduced system of equations,

$$My = \hat{p}, \tag{16}$$

where

$$M = \begin{pmatrix} (0, I)Q_1 & (-I, 0)Q_2 & & & & \\ & (0, I)Q_2 & (-I, 0)Q_3 & & & \\ & & \vdots & \vdots & & \\ & & & (0, I)Q_{k-2} & (-I, 0)Q_{k-1} & \\ & & & & (0, I)Q_{k-1} & (-I, 0)Q_k \end{pmatrix} \tag{17}$$

and

$$\hat{p} = \begin{pmatrix} p_2^{(up)} - p_1^{(low)} \\ p_3^{(up)} - p_2^{(low)} \\ \cdots \\ p_{k-1}^{(up)} - p_{k-2}^{(low)} \\ p_k^{(up)} - p_{k-1}^{(low)} \end{pmatrix}. \tag{18}$$

Note that $M$ has a dimension of $t \times (n - m + t)$. After solving system (16) for $y$, we can retrieve the solution $z_i$ via Equation (11).

We provide the following two lemmas and a theorem in order to show that the proposed algorithm finds the minimum 2-norm solution.

LEMMA 2.1. *Let $v = (v_1^T, v_2^T, \ldots, v_k^T)^T$ be a vector of size $N$ and partitioned into $k$ subvectors. Then, $||v||_2$ is minimized if and only if each $||v_i||_2$ is minimized.*

PROOF. By definition of 2-norm, $||v||_2^2 = \sum_N |v_i|^2 = ||v_1||_2^2 + ||v_2||_2^2 + \cdots + ||v_k||_2^2$. Therefore, $||v||_2$ is minimized if and only if each $||v_i||_2$ is minimized. □

LEMMA 2.2. *Assume that the minimum norm solution for the undertermined systems of equations in Equation (14) is obtained. Then, $||z_i||_2$ in Equation (11) is also minimized.*

PROOF. By Lemma 2.1, if the minimum norm solution $y$ to the reduced system is obtained, then $||y_i||_2$ is also minimized. Given $z_i = p_i + Q_i y_i$ for $i = 1, 2, \ldots, k$,

$$\begin{aligned} ||z_i||_2^2 &= ||p_i + Q_i y_i||_2^2 \\ &= (p_i + Q_i y_i)^T (p_i + Q_i y_i) \\ &= ||p_i||_2^2 + 2y_i^T Q_i^T p_i + ||y_i||_2^2 \ (Q_i^T p_i = 0 \text{ since } Q_i \text{ is orthogonal to } p_i) \\ &= ||p_i||_2^2 + ||y_i||_2^2. \end{aligned}$$

Since $||y_i||_2^2$ is minimized and $p_i$ is a fixed particular solution, $||z_i||_2$ is also minimized. □

THEOREM 2.3. *If the minimum norm solution is obtained for the reduced system in the proposed algorithm, then the algorithm obtains the minimum norm solution x of the underdetermined systems of equations.*

PROOF. If the minimum norm solution is obtained for the reduced system, then $||z_i||_2$ is minimized for each $i$ by Lemma 2.2. Therefore, $\|x\|_2$ is also minimized by Lemma 2.1 because $z_i$ subvectors constitute the solution vector $x$. □

### 2.2. Parallelization

In this subsection, we provide the details of the parallel algorithm and its implementation based on the theoretical findings given in the previous subsection. The solution process of the proposed algorithm can be summarized as follows:

(a) Obtain a particular solution $p_i$ according to Equation (13).
(b) Solve the reduced system $My = \hat{p}$ according to Equation (16).
(c) Retrieve the solution subvector $z_i$ according to Equation (11).

Stage (a) involves the solution of $k$ independent linear least squares problems, which can be done in parallel without any communication. In Stage (b), we solve a smaller underdetermined linear system, which can be done either sequentially or in parallel. We note that the solution of the reduced system can be computed either directly by forming it explicitly or iteratively without forming it explicitly, both requiring some communication. The retrieval of the solution, namely Stage (c), is done again in parallel without any communication. If needed, an optional last step involves gathering the global solution vector in one of the processors, which requires some communication.

The pseudocode of the proposed parallel algorithm for processor $i$ is given in Algorithm 1. The coefficient matrix and the right hand side vector of the given underdetermined linear system are distributed among processors by assigning each diagonal block $E_i$ and the respective subvector $f_i$ to processor $i$ for $i = 1, 2, \ldots, k$.

At line 1, each processor $i$ concurrently and independently applies QR factorization on the coefficient matrix $E_i^T$ of the local underdetermined least square problem $E_i z = f_i$. Sparse QR factorization of SuiteSparseQR [Davis 2011] package is used in local sparse QR factorization operations. For performance and storage efficiency, the orthogonal matrices generated by the local QR factorizations are stored in Householder form at line 1. At line 2, each processor $i$ concurrently computes particular solution $p_i$ according to Equation (13). In the "if-then-else" statement between lines 3 and 9, the reduced system $My = \hat{p}$ is gathered in the master processor. For this purpose, all processors except the first and the last one send subvectors $p_i^{(low)}$ and $p_i^{(up)}$ and submatrices $(0, I)Q_i$ and $(-I, 0)Q_i$ to the master processor. The last processor only needs to send subvector $p_k^{(low)}$ and submatrix $(-I, 0)Q_k$ to the master processor. Each processor $i$ obtains local $(0, I)(\hat{Q}_i, Q_i)$ and/or $(-I, 0)(\hat{Q}_i, Q_i)$ matrices by applying $(0, I)$ and $(-I, 0)$ to $(\hat{Q}_i, Q_i)$ in the Householder form for $i = 2, \ldots, k$ and $i = 1, \ldots, k-1$, respectively. Note that after

**ALGORITHM 1:** Parallel Algorithm for $k$-processor System (Pseudocode for Processor $i$)

**Input**: Block matrix $E_i$, RHS subvector $f_i$
**Output**: Solution vector $z_i$

1 Apply QR factorization $E_i^T = (\hat{Q}_i, Q_i)\begin{pmatrix} R_i \\ 0 \end{pmatrix}$
2 Obtain local particular solution $p_i = \hat{Q}_i({R_i}^{-T} f_i)$
3 **if** $1 < i < k$ **then** ▷ processor i neither master nor the last processor
4 | Send $p_i^{(low)}$, $p_i^{(up)}$, $(0, I)Q_i$ and $(-I, 0)Q_i$ to master processor (processor 1)
5 **else if** $i = k$ **then** ▷ processor is the last processor
6 | Send $p_k^{(low)}$ and $(-I, 0)Q_k$ to master processor
7 **else** ▷ master processor
8 | Receive $p_i$ subvectors and $Q_i$ submatrices
9 **endif**
10 **if** $i = 1$ **then** ▷ master Processor
11 | Construct coefficient matrix $M$ and RHS vector $\hat{p}$ for reduced system (see Equations (17) and (18))
12 | Find minimum 2-norm solution $y$ of the reduced system $My = \hat{p}$
13 | Scatter solution vector $y$ among processors so that processor $i$ receives subvector $y_i$
14 **endif**
15 Compute the solution subvector $z_i = p_i + Q_i y_i$ (see Equation 11)

forming the upper and lower parts of $Q_i$ and $\hat{Q}_i$, we discard the parts of the $\hat{Q}_i$ because we do not need it in the algorithm.

In the "if" statement between lines 10–14, the master processor first constructs the coefficient matrix $M$ and the right-hand-side vector $\hat{p}$ of the reduced system from the received submatrices and subvectors. It then finds the minimum 2-norm solution of the underdetermined linear system $My = \hat{p}$ via SuiteSparseQR_min2norm [Davis 2009] procedure with default parameters. Finally, the master processor scatters the solution vector $y$ among processors through collective communication operation *MPI_Scatterv* provided by the MPI library [Gropp et al. 1996]. At line 15, each processor $i$ concurrently computes the solution subvector $z_i$ by using $y_i$ according to Equation (11).

We should note here that the gather operation on the master processor is presented through point-to-point communications, as shown in the "if-then-else" statement between lines 3–9, for the sake of clarity of the presentation. In our implementation, this gather operation is performed using the collective communication operation *MPI_Gatherv* provided by the MPI library.

Solving the reduced system is the only sequential part in our parallel algorithm due to the limitation of the current software implementation. We have experimented with multithreaded version of SuiteSparseQR for solving the reduced system $My = \hat{p}$; however, it does not attain speedup on the solution time of the reduced system. We also have experimented with ScaLAPACK [Blackford et al. 1997] subroutine PDGELS for the parallel solution of the reduced system; however, we did not obtain speedup on more than 2 cores.

## 3. EXPERIMENTAL RESULTS

### 3.1. Datasets

Matrix collections such as UF Sparse Matrix Collection [Davis and Hu 2011] and Matrix Market [Boisvert et al. 1996] do not include many sparse matrices in a similar form of the problem defined in Section 2. For evaluating the performance of the proposed algorithm, we generated two datasets, which are referred here as realistic and synthetic datasets. The construction of realistic and synthetic datasets is described in

Table I. Properties of Realistic Test Matrix Instances

| Matrix | Diagonal Block | Coefficient Matrix | | | | Seq. Soln. Time | |
|---|---|---|---|---|---|---|---|
| ID | (*Cond.Num.*) | $t_i$ | $m$ | $n$ | $nnz$ | S.M. | D.M. |
| 1 | | 5 | 756,608 | 1,887,237 | 7,549,056 | 14.29 | 10.36 |
| 2 | | 10 | 756,608 | 1,886,922 | 7,549,056 | 14.35 | 10.35 |
| 3 | graphics | 20 | 756,608 | 1,886,292 | 7,549,056 | 14.40 | 10.41 |
| 4 | ($1.59e+8$) | 50 | 756,608 | 1,884,402 | 7,549,056 | 14.43 | 10.46 |
| 5 | | 100 | 756,608 | 1,881,252 | 7,549,056 | 14.47 | 10.49 |
| 6 | | 5 | 620,352 | 1,820,613 | 6,456,000 | 30.83 | 20.52 |
| 7 | | 10 | 620,352 | 1,820,298 | 6,456,000 | 31.44 | 20.79 |
| 8 | kemelmacher | 20 | 620,352 | 1,819,668 | 6,456,000 | 31.87 | 20.94 |
| 9 | ($2.38e+4$) | 50 | 620,352 | 1,817,778 | 6,456,000 | 31.94 | 21.08 |
| 10 | | 100 | 620,352 | 1,814,628 | 6,456,000 | 32.46 | 21.38 |
| 11 | | 5 | 705,792 | 1,709,893 | 6,555,648 | 8.18 | 5.74 |
| 12 | | 10 | 705,792 | 1,709,578 | 6,555,648 | 8.12 | 5.79 |
| 13 | psse0 | 20 | 705,792 | 1,708,948 | 6,555,648 | 8.22 | 5.81 |
| 14 | ($1.07e+6$) | 50 | 705,792 | 1,707,058 | 6,555,648 | 8.43 | 5.96 |
| 15 | | 100 | 705,792 | 1,703,908 | 6,555,648 | 8.88 | 6.32 |
| 16 | | 5 | 705,792 | 916,037 | 3,672,064 | 6.09 | 4.98 |
| 17 | | 10 | 705,792 | 915,722 | 3,672,064 | 6.29 | 5.09 |
| 18 | psse1 | 20 | 705,792 | 915,092 | 3,672,064 | 6.31 | 5.25 |
| 19 | ($1.12e+6$) | 50 | 705,792 | 913,202 | 3,672,064 | 6.84 | 5.54 |
| 20 | | 100 | 705,792 | 910,052 | 3,672,064 | 7.60 | 6.23 |
| 21 | | 5 | 705,792 | 1,832,261 | 7,376,768 | 8.83 | 6.15 |
| 22 | | 10 | 705,792 | 1,831,946 | 7,376,768 | 8.79 | 6.48 |
| 23 | psse2 | 20 | 705,792 | 1,831,316 | 7,376,768 | 9.01 | 6.31 |
| 24 | ($1.03e+6$) | 50 | 705,792 | 1,829,426 | 7,376,768 | 9.49 | 6.69 |
| 25 | | 100 | 705,792 | 1,826,276 | 7,376,768 | 10.37 | 7.25 |
| 26 | | 5 | 315,456 | 677,765 | 2,981,824 | 6.32 | 4.88 |
| 27 | | 10 | 315,456 | 677,450 | 2,981,824 | 6.62 | 4.87 |
| 28 | gemat1 | 20 | 315,456 | 676,820 | 2,981,824 | 6.66 | 4.92 |
| 29 | ($1.17e+8$) | 50 | 315,456 | 674,930 | 2,981,824 | 6.88 | 5.16 |
| 30 | | 100 | 315,456 | 671,780 | 2,981,824 | 7.31 | 5.47 |

$t_i$ : overlap size, $m$ : # of rows, $n$ : # of columns, $nnz$ : # of nonzeros.
Seq. Soln. Time: Sequential solution time (seconds) on a single core.
S.M.: Shared Memory, D.M.: Distributed Memory.

Sections 3.1.1 and 3.1.2, respectively. We also include a dataset obtained from a real-world application, as described in Section 3.1.3. In all experiments, the right-hand-side vectors of the underdetermined systems are set to a vector whose elements are all ones.

*3.1.1. Realistic Dataset.* We have created our realistic dataset by linking together 64 copies of several real rectangular matrices from the UF Sparse Matrix Collection in order to construct underdetermined linear systems of the form given in Equation (6). UF Sparse Matrix Collection has 37 rectangular matrices in least squares, computer graphics/vision, and power network problems. To illustrate the performance of parallel algorithms, we have selected full rank matrices that have more than 1,000 columns and rows. Due to the memory limitation, the matrices that have more than 50,000 columns or rows and have more than 500,000 nonzeros are excluded. After applying these criteria, six matrices (shown in Table I) remain for generating our realistic dataset.

In the construction of a coefficient matrix, if the shape of the underlying real matrix is tall and skinny, then the transpose of the matrix is used in order to construct an

Table II. Properties of Synthetic Test Matrices

| Matrix | Diagonal Block | | | Coefficient Matrix | | | | Seq. Soln. Time | |
|---|---|---|---|---|---|---|---|---|---|
| ID | $m_i$ | $\tilde{n}_i$ | $nnz/m_i$ | $t_i$ | $m$ | $n$ | $nnz$ | S.M. | D.M. |
| 1 | 4,014.45 | 5,013.11 | 10.02 | 5 | 256,925 | 320,466 | 2,573,387 | 256.68 | 158.30 |
| 2 | 4,012.75 | 5,011.20 | 9.99 | 10 | 256,816 | 320,151 | 2,565,271 | 263.36 | 161.79 |
| 3 | 3,984.64 | 5,002.05 | 10.00 | 20 | 255,017 | 318,871 | 2,549,146 | 261.81 | 164.05 |
| 4 | 4,003.02 | 5,006.02 | 9.97 | 50 | 256,193 | 317,235 | 2,554,582 | 289.03 | 179.56 |
| 5 | 3,979.41 | 5,003.92 | 10.00 | 100 | 254,682 | 313,951 | 2,546,786 | 338.93 | 202.36 |
| 6 | 4,000.36 | 5,011.11 | 20.01 | 5 | 256,023 | 320,396 | 5,122,404 | 501.07 | 320.49 |
| 7 | 4,007.44 | 5,000.70 | 19.97 | 10 | 256,476 | 319,415 | 5,120,651 | 510.48 | 328.27 |
| 8 | 3,995.78 | 4,995.19 | 19.94 | 20 | 255,730 | 318,432 | 5,100,235 | 536.83 | 343.52 |
| 9 | 4,007.44 | 5,000.56 | 19.97 | 50 | 256,476 | 316,886 | 5,120,651 | 607.22 | 392.03 |
| 10 | 4,003.39 | 5,008.63 | 20.00 | 100 | 256,217 | 314,252 | 5,123,315 | 783.57 | 499.13 |
| 11 | 4,010.45 | 4,998.69 | 29.91 | 5 | 256,669 | 319,601 | 7,676,243 | 652.17 | 428.48 |
| 12 | 4,000.47 | 4,993.03 | 29.87 | 10 | 256,030 | 318,924 | 7,648,085 | 667.33 | 436.80 |
| 13 | 3,997.77 | 4,995.28 | 29.89 | 20 | 255,857 | 318,438 | 7,646,384 | 723.38 | 470.64 |
| 14 | 4,020.23 | 5,002.38 | 29.93 | 50 | 257,295 | 317,002 | 7,700,351 | 872.08 | 567.53 |
| 15 | 3,998.67 | 5,011.69 | 30.00 | 100 | 255,915 | 314,448 | 7,674,427 | 1,038.83 | 798.59 |

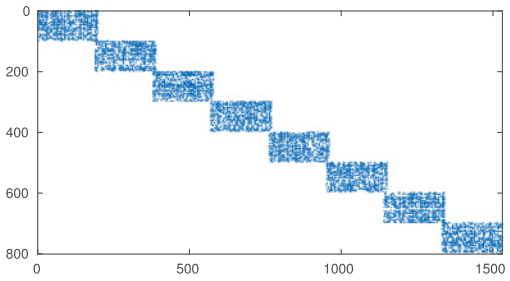
$m_i$ : average # of rows, $\tilde{n}_i$ : average # of columns, $nnz/m_i$ : average of # nonzeros per row.
$t_i$ : overlap size, $m$ : # of rows, $n$ : # of columns, $nnz$ : # of nonzeros.
Seq. Soln. Time: sequential solution time (seconds) on a single core.
S.M.: Shared Memory, D.M.: Distributed Memory.

underdetermined linear system. Each of the 64 diagonal blocks (i.e., $E_i$ blocks) of a coefficient matrix is obtained by slightly perturbing nonzero values of the underlying real matrix in a random manner. That is, diagonal blocks differ slightly only in nonzero values, whereas they have the same sparsity pattern. For each nonzero, the perturbation amount is randomly selected in the range $[-\alpha \times \eta, \alpha \times \eta]$, where $\eta = \|A\|_F$ and $\alpha = 0.10$ in order not to deviate too much from the real matrix. The original condition numbers of the diagonal block matrices are shown in Table I.

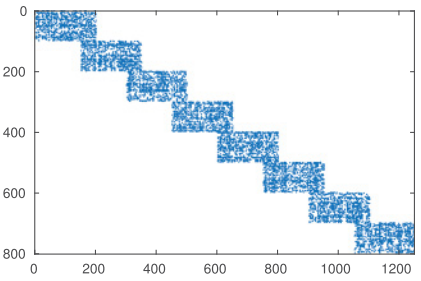
As discussed in Section 2.2, sizes of the column overlaps between successive blocks define the size of the reduced system $M$, which is the bottleneck of our parallel algorithm. In order to observe the effects of overlap sizes on the performance of the proposed parallel algorithm, we generated five coefficient matrices with different amounts of column overlaps of $t_i = 5, 10, 20, 50,$ and 100 for each underlying real matrix. Thus, the realistic dataset contains 30 underdetermined coefficient matrices. The properties of these coefficient matrices are shown in Table I. Note that each of the five coefficient matrices obtained from the same underlying real matrix have the same number of rows and nonzeros, whereas they have slightly different number of columns because of different column overlap sizes.

*3.1.2. Synthetic Dataset.* We have constructed a synthetic dataset of 15 underdetermined systems of the form given in Equation (6) from sparse uniformly distributed random diagonal blocks. In the constructed coefficient matrices, each diagonal block $E_i$ has more columns than rows. We produce problems that are as large as possible and can fit into the memory when solving the minimum norm solution on our test platforms. In the synthetic dataset, the coefficient matrices have different dimensions, nonzero densities and overlap sizes. The properties of the synthetic underdetermined systems are shown in the Table II.

Each synthetic test matrix contains 64 diagonal blocks and they are constructed by using average block diagonal size of $m_i \times \tilde{n}_i = 4{,}000 \times 5{,}000$. Both row and column dimensions of diagonal blocks are perturbed by a maximum amount of 5% on the preceding given averages. Each diagonal block $E_i$ has a condition number of




(a) $A_a = 800 \times 1{,}530$ with overlap size of 10




(b) $A_b = 800 \times 1{,}250$ with overlap size of 50

Fig. 1. Two sample matrices with eight sparse diagonal blocks of size $100 \times 200$.

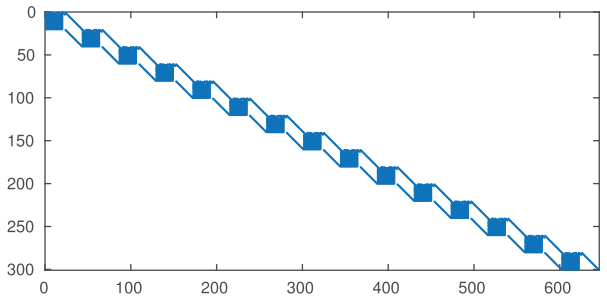



Fig. 2. Nonzero pattern of `grow15`.

approximately $10^2$ and a nonzero density of each diagonal block is set so that each row has about 10, 20, and 30 nonzeros by using *sprand* procedure in MATLAB version 2008. Test matrices with different amounts of column overlaps of $t_i = 5, 10, 20, 50$, and 100 are generated as in the realistic dataset.

Two sample matrices produced by our synthetic matrix generation tool are shown in Figure 1. These matrices have 8 rectangular sparse diagonal blocks of size $100 \times 200$ that have 10 nonzeros per row. In Figure 1(a), matrix $A_a$ has a column overlap size of 10 and hence has a size of $800 \times 1{,}530$. In Figure 1(b), matrix $A_b$ has a column overlap size of 50 and hence has a size of $800 \times 1{,}250$.

*3.1.3. Real-World Dataset.* Staircase-structured linear programming problems may arise from multistage or time-phased systems such as structural design optimization [Ho 1975] and economic planning over time [Fourer 1982]. There are methods that utilize Newton's approach for solving linear programming problems [Han 1980], quadratic programming problems [Bramley and Winnicka 1996] as well as linear feasibility problems [Bramley and Winnicka 1996; Pinar 1998; Dax 2008]. At each Newton iteration, the minimum norm solution of an underdetermined linear system of equations needs to be obtained. Thus, solving such problems with staircase constraint matrices using Newton-based approaches constitute a real-world application of obtaining the minimum norm solution of an underdetermined linear least squares problem in which the coefficient matrix has a block diagonal column overlapping form.

One example of the staircase-structured linear programming problem is `grow15` [Fourer 1983], which models the input-output analysis of the U.S. economy [Glassey and Benenson 1975] with 15 time periods. Figure 2 shows the nonzero pattern of the coefficient matrix of `grow15`. This matrix has 15 diagonal blocks, and each block consists of 20 rows and 63 columns (except the first block, which has 43 columns)

Table III. Properties of Real-world Matrix Instances

| Matrix | Coefficient Matrix | | | | | Seq. Soln. Time | |
|---|---|---|---|---|---|---|---|
| ID | Name | $t_i$ | $m$ | $n$ | $nnz$ | S.M. | D.M. |
| 1 | grow2560 | 20 | 51,200 | 110,080 | 962,540 | 1.45 | 1.20 |
| 2 | grow5120 | 20 | 102,400 | 220,160 | 1,925,100 | 2.99 | 2.43 |
| 3 | grow10240 | 20 | 204,800 | 440,320 | 3,85,0220 | 6.22 | 4.98 |
| 4 | grow20480 | 20 | 409,600 | 880,640 | 7,700,460 | 13.29 | 10.30 |

$t_i$ : overlap size, $m$ : # of rows, $n$ : # of columns, $nnz$ : # of nonzeros.
Seq. Soln. Time: Sequential solution time (seconds) on a single core.
S.M.: Shared Memory, D.M.: Distributed Memory.

and successive diagonal blocks have 20 overlapping columns. In order to observe the parallel scalability of the proposed algorithm, data of the same input-output analysis with 2,560, 5,120, 10,240 and 20,480 time periods were used. In the corresponding coefficient matrices, the number of periods is equal to the number of diagonal blocks and the amount of column overlap is $t_i = 20$. Properties of the real-world dataset is shown in Table III.

### 3.2. Experimental Framework

The performance of the proposed algorithm is tested on both shared and distributed memory architectures. As the shared memory architecture, we use a single-node 64-core computer that contains 4 AMD Opteron 6376 processors, with each processor having 16 cores running at 2.3GHz and a total of 128GB of DDR3 memory. As the distributed memory architecture, we use the SuperMUC (phase 1 thin nodes) [GCS Supercomputer 2012] system. SuperMUC is an IBM System x iDataPlex dx360M4 system that consists of 9,216 nodes, where each node contains two 8-core Intel(R) Xeon(R) E5-2680 processors running at 2.7GHz and 32GB RAM. Nodes are interconnected with a high-bandwidth low-latency switch network (Infiniband FDR10).

For comparing the performance of the proposed algorithm, there are not any existing QR factorization routines specifically designed for matrices that have overlapping sparse diagonal blocks. Hence, as the baseline algorithm, we use SuiteSparseQR [Davis 2011], which is a multithreaded multifrontal general sparse QR factorization procedure in SuiteSparse [Davis 2013] software package. SuiteSparseQR uses Intel's Threading Building Blocks (TBB) [Reinders 2007] library for providing parallelism on shared memory multicore architectures. METIS [Karypis and Kumar 2009] reordering technique is used for both sequential and parallel SuiteSparseQR as recommended in Davis [2009]. We report sequential running times of the test matrices in Tables I, II, and III on two different computing platforms using sequential SuiteSparseQR.

In Davis [2011], the best performance results for SuiteSparseQR are achieved by using a mixture of TBB threads and multithreaded BLAS. We have conducted experiments to see how the performance of SuiteSparseQR varies with using mixture of TBB threads and multithreaded BLAS on our datasets. In experiments for the real-world and realistic datasets, adding BLAS parallelism does not improve the performance, so the best performance is achieved via using only TBB threads for all of the 34 matrix instances. For the synthetic dataset, using multiple BLAS threads improves the performance on some matrix instances. Table IV displays the best settings according to average speedup values obtained for $1, 2, \ldots, c$ TBB threads and $1, 2, \ldots, c$ BLAS threads using $c$ cores of the node of the distributed memory architecture and the shared memory architecture for $c = 2, 4, 8, 16$ and $c = 2, 4, 8, 16, 32, 64$, respectively. The parameters for SuiteSparseQR are selected as advised in Davis [2009], *SPQR_nthreads* is set to the number of cores, *SPQR_grain* is set to 2 times the number of cores and *SPQR_small* is set to $10^6$.

Table IV. Settings for Parallel SuiteSparseQR

| | | 2 cores | | 4 cores | | 8 cores | | 16 cores | | 32 cores | | 64 cores | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| System | Dataset | T | B | T | B | T | B | T | B | T | B | T | B |
| Shared Memory | Realistic | 2 | 1 | 4 | 1 | 8 | 1 | 16 | 1 | 32 | 1 | 64 | 1 |
| | Synthetic | 2 | 1 | 4 | 1 | 8 | 1 | 8 | 2 | 16 | 2 | 64 | 1 |
| | Real-World | 2 | 1 | 4 | 1 | 8 | 1 | 16 | 1 | 32 | 1 | 64 | 1 |
| Distributed Memory | Realistic | 2 | 1 | 4 | 1 | 8 | 1 | 16 | 1 | - | - | - | - |
| | Synthetic | 2 | 1 | 2 | 2 | 8 | 2 | 16 | 4 | - | - | - | - |
| | Real-World | 2 | 1 | 4 | 1 | 8 | 1 | 16 | 1 | - | - | - | - |

T: Number of TBB threads, B: Number of BLAS threads.

Intel compiler versions 14.0.1 and 15.0 with Intel Math Kernel Library (MKL) versions 11.1 and 11.2 are used on the shared and distributed memory architectures, respectively. The proposed algorithm is implemented in C++ programming language. MPICH version 3.1.4 and IBM Platform MPI version 1.4 implementations of message passing interfaces are used on the shared and distributed memory architectures, respectively. In the proposed algorithm, sequential SuiteSparseQR with single-threaded BLAS is used for local QR factorization in each block of the proposed algorithm, and each MPI process is mapped to one core on the shared and distributed memory architectures. Note that the performance of the proposed method may increase by adding the parallelism mechanisms of SuiteSparseQR and BLAS for the local QR factorizations. However, we have not observed any significant performance improvements for the datasets we used.

In the proposed algorithm, when we use a smaller number of cores than the number of diagonal blocks, each core works on multiple successive blocks.

### 3.3. Scalability Results

In this subsection, we report the results of strong scalability tests performed for both the proposed and the baseline algorithms. Figures 3, 4, and 5 display the scalability results for realistic, synthetic, and real-world datasets, respectively. All three figures depict the results as speedup curves on both shared and distributed memory computing platforms. In Figures 3 and 4, the speedup curve for each of the realistic and synthetic test matrices is obtained by averaging the speedup values at each core count over five different column overlap sizes. That is, Figures 3 and 4 effectively display average speedup curves.

In Figures 3, 4, and 5, speedup values are displayed for 2, 4, 8, 16, 32, and 64 cores of the shared memory architecture for both the proposed and baseline algorithms. For the distributed memory architecture, since the test matrices in realistic and synthetic datasets have 64 blocks, experiments for the proposed algorithm are done with at most 64 cores across four distributed nodes. Thus, Figures 3, 4, and 5 display the speedup values for 2, 4, 8, 16, 32, and 64 cores for the proposed algorithm on the distributed memory architecture. However, all three figures do not display the performance of the baseline algorithm for 32 and 64 cores on the distributed memory architecture because SuiteSparseQR can only utilize thread-level parallelism. Therefore, since the realistic dataset contains 30 test matrices, Figure 3 compares the speedup results of the proposed algorithm against the baseline algorithm for $30 \times 6 = 180$ and $30 \times 4 = 120$ parallel running instances on the shared and distributed memory architectures, respectively. Similarly, since the synthetic dataset contains 15 test matrices, Figure 4 compares the speedup results of the proposed algorithm against the baseline algorithm for $15 \times 6 = 90$ and $15 \times 4 = 60$ parallel running instances on the shared and distributed memory architectures, respectively. Since the real-world dataset contains

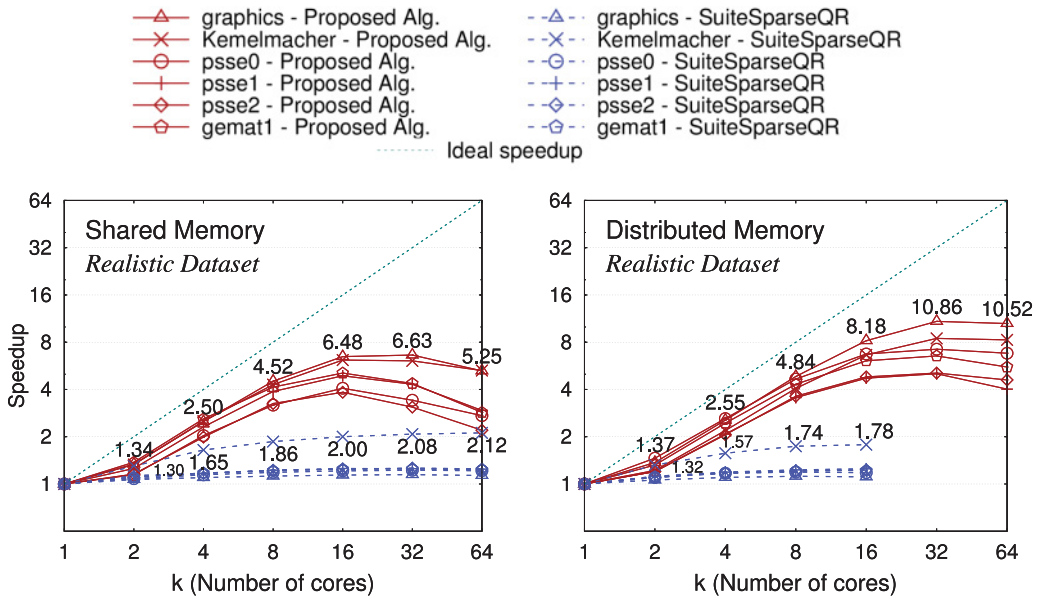



Fig. 3. Average speedup curves for realistic matrices (averages over five different overlap sizes).

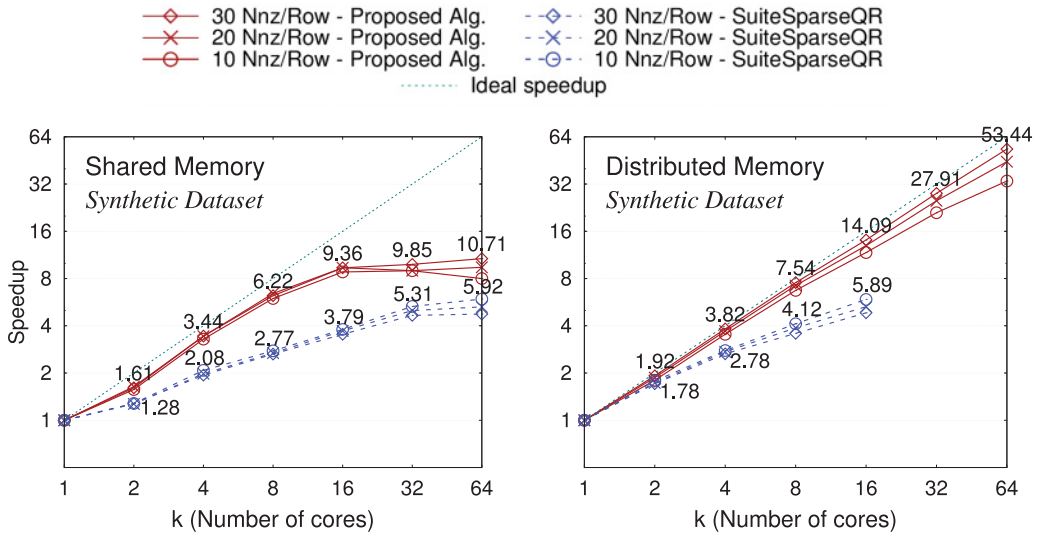



Fig. 4. Average speedup curves for synthetic matrices (averages over five different overlap sizes).

four matrices, Figure 5 compares the speedup results of the proposed algorithm against the baseline algorithm for $4 \times 6 = 24$ and $4 \times 4 = 16$ parallel running instances on the shared and distributed memory architectures, respectively.

In Figures 3, 4, and 5, solid lines (red color) and dashed lines (blue color) are respectively used to show the speedup curves for the proposed algorithm and SuiteSparseQR. These figures also show the maximum average speedup values attained by both proposed and baseline algorithms on each core count for the sake of a better performance comparison.

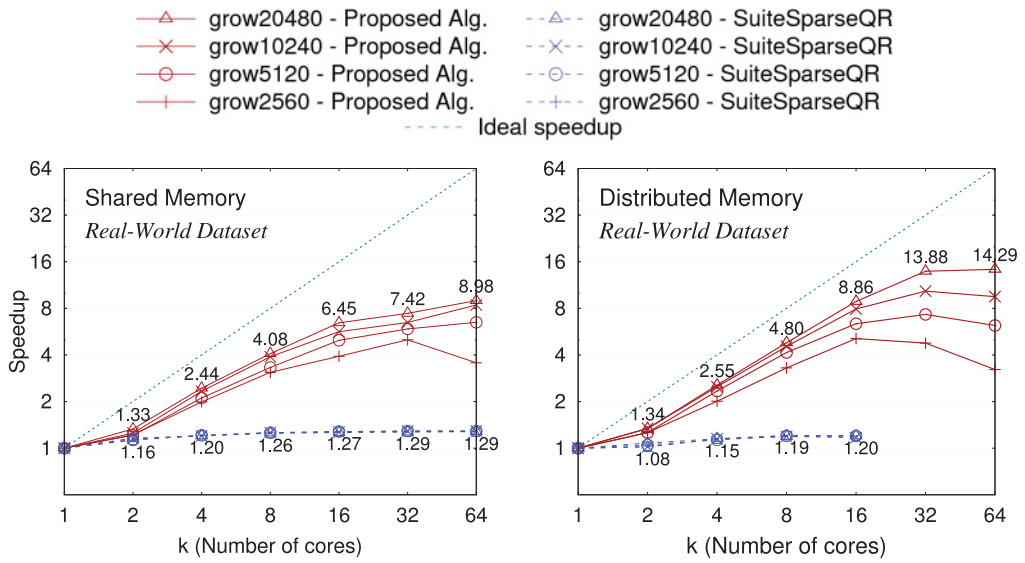



Fig. 5. Speedup curves of real-world matrices.

*3.3.1. Realistic Dataset.* As shown in Figure 3, the proposed algorithm shows much better scalability than the baseline algorithm for the realistic dataset. The proposed algorithm achieves considerably better speedup values than the baseline algorithm on 275 out of 300 parallel running instances. The remaining 25 running instances, for which the proposed algorithm fails to achieve better performance, are on 2 cores (15 out of 25 instances) and are for the matrices with the largest column overlap size of 100 on 32 and 64 cores (10 out of 25 instances).

As shown in Figure 3, the performance gap between the proposed and baseline algorithms increases considerably with increasing core counts until 16 cores for the shared memory architecture and until 64 cores for the distributed memory architecture in favor of the proposed algorithm, whereas the performance gap slightly decreases on 32 and 64 cores for the shared memory architecture. This is because the memory bandwidth begins to become a bottleneck for the proposed algorithm on high number of cores in the shared memory architecture, since it uses relatively more memory than the baseline algorithm. As shown in the figure, the proposed algorithm achieves best average speedups of 6.63 and 10.86 for the `graphics`-based matrices on the shared and distributed memory architectures, respectively. The baseline algorithm achieves the best average speedups of 2.12 and 1.78 for the `Kemelmacher`-based matrices on the shared and distributed memory architectures, respectively.

*3.3.2. Synthetic Dataset.* As shown in Figure 4, the proposed algorithm also scales much better than the baseline algorithm for the synthetic dataset. The proposed algorithm achieves considerably better speedup values than the baseline algorithm on almost all of the 150 parallel running instances and fails to achieve better performance only for the 2 running instances that are on 2 cores of the distributed memory architecture.

As shown in Figure 4, for the synthetic dataset, the performance of the proposed algorithm slightly decreases for sparser coefficient matrices. This is because, for a given number of processors, with decreasing matrix density, the parallelization overhead due to communication and sequential solution of the reduced system remains almost the same, whereas the amount of concurrent local computations decrease.

As shown in Figure 4, the proposed algorithm scales up until 32 and 64 cores on the shared and distributed memory architectures, respectively. The proposed algorithm achieves maximum average speedup of 10.71 and 53.44 for matrices with 30 nonzeros per row on 64 cores of the shared and distributed memory architectures, respectively. On the other hand, the baseline algorithm achieves a maximum average speedup of only 5.92 and 5.89 for the matrices with 10 nonzeros per row on 64 and 16 cores of the shared and distributed memory architectures, respectively.

*3.3.3. Comparison of Realistic and Synthetic Datasets.* A comparison of Figures 3 and 4 shows that both proposed and baseline algorithms achieve much better speedup for the synthetic dataset. This experimental finding is due to the fact that the problem sizes are much larger for synthetic dataset compared to those for the realistic dataset. Note that the problem size refers to the amount of work involved in the sequential algorithm, which is also indicated in the sequential running times in the last two columns of Tables I and II.

*3.3.4. Effects of Overlap Size.* Figure 6 is given in order to show the effect of overlap size on the performance of the proposed and baseline algorithms for both realistic and synthetic datasets. In the figure, the speedup value for each overlap size is depicted as the average of the speedups obtained for six realistic or three synthetic test matrices (varied nonzero densities of 10, 20, and 30nnz/row) on the given core count.

As shown in Figure 6, for both realistic and synthetic datasets, the performance of the proposed algorithm decreases considerably with increasing overlap size, whereas the performance of the baseline algorithm increases slightly with increasing overlap size. The former experimental finding is expected because increasing overlap size increases the size of the reduced system and thus increases the sequential portion of the proposed parallel algorithm. The latter experimental finding is because of the fact that the performance of the baseline algorithm is not sensitive to the overlap size and the column size of the coefficient matrix decreases with increasing overlap size in our experimental setting. Despite these expected experimental findings, the proposed algorithm achieves better average speedup than the baseline algorithm for all parallel running instances except for the realistic matrices with largest overlap size of 100 on 64 cores of the shared memory architecture.

In accordance with the previous discussion, the proposed algorithm achieves maximum speedup values for the matrices with smallest column overlap size of 5, whereas the baseline algorithm achieves maximum speed values for the matrices with the largest overlap size of 100. As shown in Figure 6, on the shared memory architecture, the proposed algorithm achieves the maximum average speedup of 8.40 and 11.08 on 32 cores for realistic and synthetic datasets, respectively, whereas the baseline algorithm obtains maximum speedups of 1.39 and 6.02 on 64 cores for realistic and synthetic datasets, respectively. On the distributed memory architecture, the proposed algorithm obtains maximum speedups of 16.47 and 54.10 on 64 cores for the realistic and synthetic datasets, respectively, whereas the baseline algorithm achieves speedups of only 1.31 and 6.23 on 16 cores for the realistic and synthetic datasets, respectively.

*3.3.5. Real-World Dataset.* Figure 5 depicts a much better parallel scalability for the proposed algorithm compared to the baseline algorithm for the real-world dataset. The scalability of the proposed algorithm improves as the problem size gets larger. Hereby, the proposed algorithm scales up until 64 cores for the largest problem (`grow20480`) on both architectures.

As shown in Figure 5, the proposed algorithm achieves significantly better speedup values for all test instances. The proposed algorithm achieves a speedup of 8.98 and 14.29 for `grow20480` on 64 cores of the shared and distributed memory architectures,

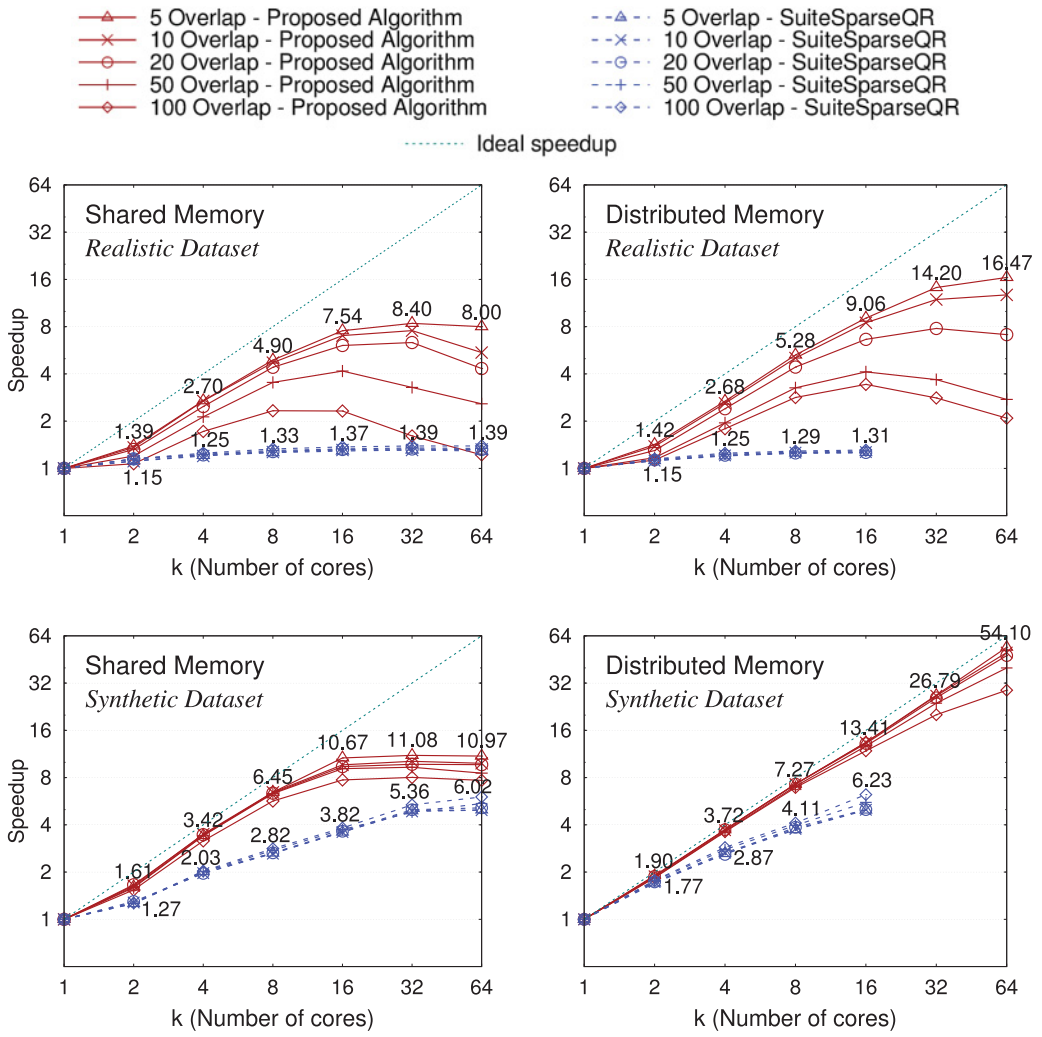



Fig. 6. Variation of average speedup curves with varying overlap sizes.

respectively. On the other hand, the baseline algorithm gives almost the same speedup of 1.29 and 1.20 for all matrices on 64 and 16 cores of the shared and distributed memory architectures, respectively.

### 3.4. Accuracy of the Numerical Results

In the earlier subsection, we studied the parallel scalability of our algorithm on two different platforms compared to the baseline algorithm. In this subsection, we will look into the numerical accuracy of the results that were obtained using the same datasets given earlier. As the exact solution is not known, we will closely examine the relative residuals and the norm of the solution vectors obtained.

Figure 7 shows relative residual norms $\Gamma_u$ of the underdetermined linear systems using the sequential SuiteSparseQR and the proposed algorithm for 2, 4, 8, 16, 32, and 64 cores for the realistic, synthetic, and real-world datasets. Relative residual norm $\Gamma_u$

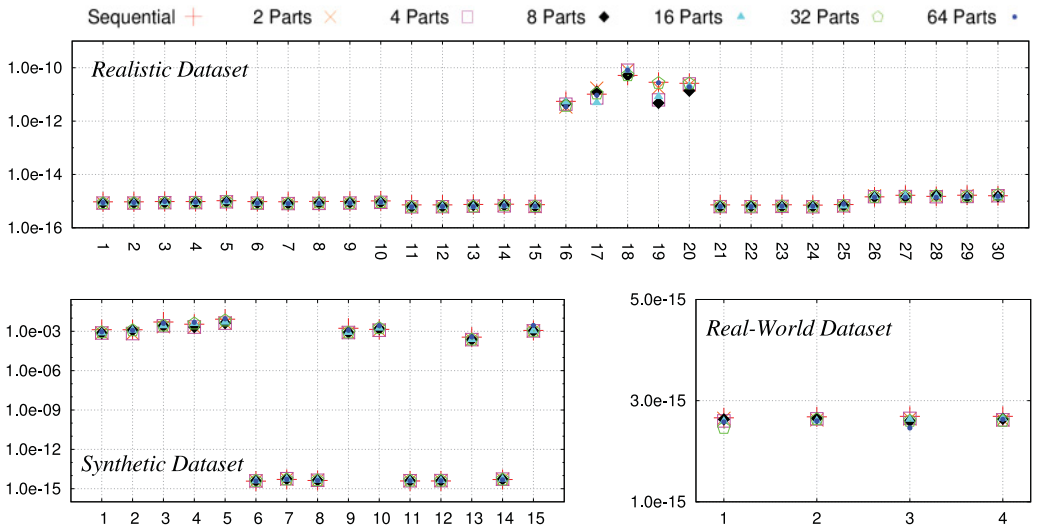



Fig. 7. Relative residual norms of sequential and the proposed parallel algorithm for the datasets.

is calculated as follows:

$$\Gamma_u = \frac{\|b - Ax\|_2}{\|b\|_2}. \tag{19}$$

The proposed algorithm produces very similar relative residuals with respect to the sequential baseline algorithm for almost all cases. In a few test instances only, the relative residuals differ slightly. We believe the reason for this is that the condition number of the matrix is always greater than the condition number of diagonal blocks. For instance, when using 64 cores, each core applies the QR factorization on just one block that has the smallest condition number. When a smaller number of cores is used, each core works on more than one block, which have a much larger condition number than those of the constituent diagonal blocks. Furthermore, matrices that have larger overlap sizes are also likely to have larger condition numbers compared to ones that have smaller overlap sizes. This implies that the sequential algorithm is expected to give slightly larger relative residual norm compared to the proposed parallel algorithm.

Figure 8 illustrates the 2-norm of the solution vectors (i.e., $\|x\|_2$) obtained using the sequential algorithm and the proposed parallel algorithm for 2, 4, 8, 16, 32, and 64 cores. As shown in the figure, the proposed algorithm finds a solution that agrees with the solution vector obtained by the sequential QR factorization.

In summary, the results show that the proposed algorithm is not only more scalable but also gives accurate results comparable to the existing algorithm based on the QR factorization. In Section 4, we will further study the accuracy of the proposed algorithm.

## 4. FORWARD AND BACKWARD ERRORS

In this section, we further look into the numerical behavior of the proposed scheme compared to the Q and seminormal equations (SNE) methods. Norm-wise relative errors of both methods are bounded by $c\kappa_2(A)u$ [Demmel and Higham 1993], where $c$ is a constant, $u$ is the unit roundoff error, and $\kappa_2(A)$ is defined as

$$\kappa_2(A) = \|A^+\|_2\|A\|_2. \tag{20}$$

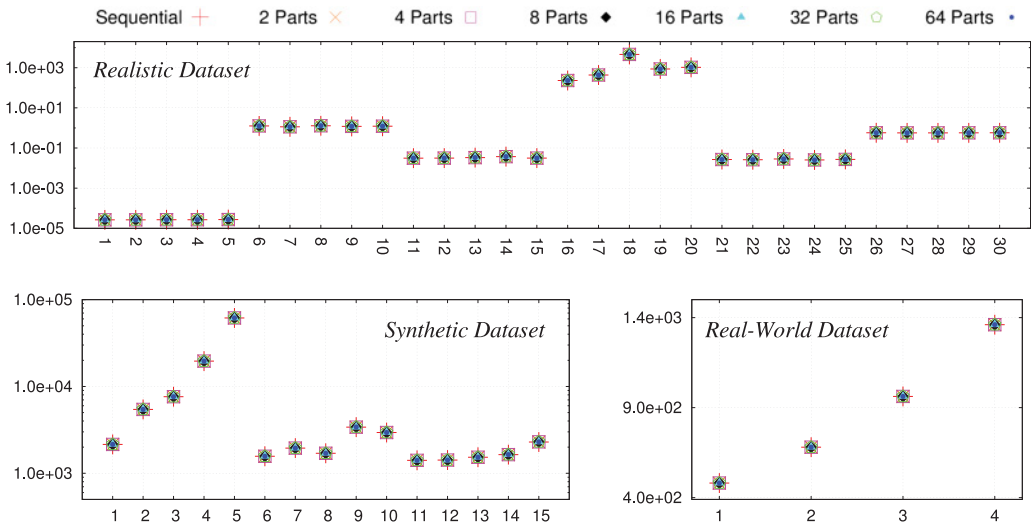



Fig. 8. Sequential and parallel $\|x\|_2$ values for the datasets.

in Demmel and Higham [1993], tighter error bounds for underdetermined linear systems are introduced. In their work, the term $\kappa_2(A)$ is replaced by

$$cond_2(A) = \||A^+||A|\|_2, \tag{21}$$

where

$$A^+ = A^T(AA^T)^{-1} \tag{22}$$

is the pseudoinverse of $A$. They note that $cond_2(A)$ could be arbitrarily smaller but could not be much bigger than $\kappa_2(A)$, and the component-wise condition number is defined as

$$cond_2(A, x) = (\||I - A^+A||A^T||A^{+^T}x|\|_2 + \||A^+|(|b| + |A||x|)\|_2)/\|x\|_2. \tag{23}$$

We have ran numerical experiments in MATLAB, in which the *double precision* and *single precision* have a unit roundoff $u \approx 2.2 \times 10^{-16}$ and $u \approx 1.2 \times 10^{-7}$, respectively. Different from simulating single-precision arithmetic in Demmel and Higham [1993], we use true *single-precision* arithmetic in our experiments. This gave us a chance to verify the results of Demmel and Higham [1993] by conducting experiments with *single*-precision arithmetic rather than simulating single precision. For forward error analysis, we regard the solution with double precision as the exact solution in our experiments.

The coefficient matrices in Table V are random matrices whose sizes are $200 \times 395$ with cascaded two dense blocks size of $100 \times 200$ with a column overlap size of 5. In each matrix, the block matrices were generated using the *randsvd* [Higham 1991] routine. Singular values ($\sigma_i$) of block matrices were distributed geometrically, that is, $\sigma_i = \alpha^{(1-i)/(n-1)}$, $i = 1 \ldots n$, where $\alpha = \kappa_2(A)$ is a parameter [Higham 2002] of *randsvd*. Entries of the right-hand sides, $b$ vectors, were normally distributed between (0, 1).

We define

$$\gamma_2(\hat{x}) = \frac{\|\hat{x} - x\|_2}{\|x\|_2}, \tag{24}$$

Table V. Forward and Backward Errors Using Three Different Algorithms for Solving Problems with Different Condition Numbers

| Mtx | Metrics | Soln. Method | $p^N(\hat{x})$ | $p^R(\hat{x})$ | $p^C(\hat{x})$ | $\gamma_2(\hat{x})$ |
|---|---|---|---|---|---|---|
| | $\kappa_2(A) = 1.03e4$ | QR | 8.52e-10 | 1.61e-10 | 5.64e-08 | 2.33e-04 |
| 1 | $cond_2(A) = 4.05e4$ | SNE | 5.11e-06 | 9.62e-07 | 3.66e-04 | 5.21e-01 |
| | $cond_2(A, x) = 9.61e4$ | Proposed Alg. | 6.84e-10 | 1.29e-10 | 4.52e-08 | 4.21e-02 |
| | $\kappa_2(A) = 1.02e5$ | QR | 7.86e-10 | 1.63e-10 | 5.62e-08 | 1.92e-03 |
| 2 | $cond_2(A) = 3.31e5$ | SNE | 1.11e-05 | 2.30e-06 | 5.32e-04 | 1.50e+01 |
| | $cond_2(A, x) = 1.72e6$ | Proposed Alg. | 1.15e-09 | 2.37e-10 | 8.15e-08 | 5.70e-02 |
| | $\kappa_2(A) = 1.05e6$ | QR | 5.82e-10 | 1.50e-10 | 5.63e-08 | 1.73e-02 |
| 3 | $cond_2(A) = 2.94e6$ | SNE | 2.11e-05 | 5.46e-06 | 1.91e-03 | 1.00e+00 |
| | $cond_2(A, x) = 8.37e10$ | Proposed Alg. | 6.97e-10 | 1.80e-10 | 6.73e-08 | 4.37e-02 |

where $\hat{x}$ is the computed solution. Three relative residuals [Demmel and Higham 1993] are

$$p^X(\hat{x}) = \max_i \frac{|b - A\hat{x}|_i}{(E_X|\hat{x}| + f_X)_i}, \qquad X = N, R, C \tag{25}$$

where $E_X$ and $f_X$ are

$$\begin{aligned} \text{norm-wise} &: E_N = \|A\|_2 e_m e_n^T, & f_N &= \|b\|_2 e_m, \\ \text{row-wise} &: E_R = |A| e_m e_n^T, & f_R &= |b|, \\ \text{component-wise} &: E_C = |A|, & f_C &= |b|. \end{aligned} \tag{26}$$

Note that in Equation (26), $e_n = (1, 1, \ldots, 1)^T \in \mathbb{R}^n$. In the SNE method, iterative refinement was not used.

In Table V, the error bounds of the work [Demmel and Higham 1993] are confirmed for SNE and Q methods. As given in Demmel and Higham [1993], the error is estimated precisely within an order of magnitude by the equation

$$\gamma_2(\hat{x}) = \frac{\|\hat{x} - x\|_2}{\|x\|_2} \approx cond_2(A)u. \tag{27}$$

The results also show that for the given problems, the proposed algorithm has better norm-wise relative error and relative residuals than the SNE method. Compared to the Q method, the proposed algorithm gives comparable relative residuals for all test problems. The norm-wise relative errors are comparable to the Q method for the matrices with larger condition numbers.

## 5. CONCLUSION

A new parallel algorithm was proposed and implemented for computing the minimum norm solution of sparse block diagonal column overlapped underdetermined systems. The proposed algorithm exploits the special structure of the coefficient matrix by handling the block diagonals independently and concurrently. Parallel scalability of the proposed scheme was shown on two different parallel architectures: a shared memory (multicore) architecture and a distributed memory (cluster) architecture for realistic, synthetic, and real-world datasets. The realistic, synthetic, and real-world datasets include 30, 15, and 4 test matrix instances, respectively. In the best case, the proposed algorithm achieves a speedup of 54.1 on 64 cores. Scalability performance of the proposed algorithm degrades with increasing column overlap size as expected. Experimental results on all datasets show the validity of the proposed algorithm on shared

memory and distributed memory architectures. Furthermore, numerical stability of the proposed scheme was studied. The proposed algorithm can be also considered as an scalable extension of any multithreaded general sparse QR factorization algorithm to distributed memory architectures for computing minimum 2-norm solution of underdetermined linear least squares problems.

## REFERENCES


Patrick R. Amestoy, Iain S. Duff, and Chiara Puglisi. 1996. Multifrontal QR factorization in a multiprocessor environment. *Numerical Linear Algebra with Applications* 3, 4 (1996), 275–300.

Cleve Ashcraft and Roger G. Grimes. 1999. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing*. SIAM.

Åke Björck. 1996. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia.

L. Susan Blackford, Jaeyoung Choi, Andy Cleary, Eduardo D'Azevedo, James Demmel, Inderjit Dhillon, Jack Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, and others. 1997. *ScaLAPACK Users' Guide*. Vol. 4. SIAM.

Ronald F. Boisvert, Roldan Pozo, Karin A. Remington, Richard F. Barrett, and Jack Dongarra. 1996. Matrix market: A web resource for test matrix collections. In *Quality of Numerical Software*. 125–137.

Randall Bramley and B. Winnicka. 1996. Solving linear inequalities in a least squares sense. *SIAM Journal on Scientific Computing* 17, 1 (1996), 275–286. DOI: http://dx.doi.org/10.1137/0917020

Alfred Bruckstein, David Donoho, and Michael Elad. 2009. From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM Review* 51, 1 (2009), 34–81. DOI: http://dx.doi.org/10.1137/060657704

Alfredo Buttari. 2013. Fine-grained multithreading for the multifrontal QR factorization of sparse matrices. *SIAM Journal on Scientific Computing* 35, 4 (2013), C323–C345.

Shane F. Cotter, Bhaskar D. Rao, Kjersti Engan, and Kenneth Kreutz-Delgado. 2005. Sparse solutions to linear inverse problems with multiple measurement vectors. *Signal Processing, IEEE Transactions on* 53, 7 (2005), 2477–2488.

Timothy A. Davis. 2009. Users guide for SuiteSparseQR, a multifrontal multithreaded sparse QR factorization package. http://faculty.cse.tamu.edu/davis/suitesparse.html.

Timothy A. Davis. 2011. Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 8.

Timothy A. Davis. 2013. SuiteSparse packages, released 4.2.1. http://faculty.cse.tamu.edu/davis/suitesparse.html.

Timothy A. Davis and Yifan Hu. 2011. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1.

Achiya Dax. 2008. A hybrid algorithm for solving linear inequalities in a least squares sense. *Numerical Algorithms* 50, 2 (2008), 97–114. DOI: http://dx.doi.org/10.1007/s11075-008-9218-3

James W. Demmel and Nicholas J. Higham. 1993. Improved error bounds for underdetermined system solvers. *SIAM Journal on Matrix Analysis and Applications* 14, 1 (1993), 1–14.

Robert Fourer. 1982. Solving staircase linear programs by the simplex method, 1: Inversion. *Mathematical Programming* 23, 1 (1982), 274–313.

Robert Fourer. 1983. Solving staircase linear programs by the simplex method, 2: Pricing. *Mathematical Programming* 25, 3 (1983), 251–292.

Robert Fourer. 1984. Staircase matrices and systems. *SIAM Review* 26, 1 (1984), 1–70.

Leibniz Supercomputing Centre GCS Supercomputer. 2012. SuperMUC Petascale System. Retrieved from www.lrz.de.

Philip E. Gill and Walter Murray. 1973. A numerically stable form of the simplex algorithm. *Linear Algebra and Its Applications* 7, 2 (1973), 99–138.

C. Roger Glassey and Peter Benenson. 1975. *Quadratic Programming Analysis of Energy in the United States Economy. Final Report*. Technical Report. University of California, Berkeley.

Gene Golub, Ahmed H. Sameh, and Vivek Sarin. 2001. A parallel balance scheme for banded linear systems. *Numerical Linear Algebra with Applications* 8, 5 (2001), 285–299.

William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing* 22, 6 (1996), 789–828.

Shih-Ping Han. 1980. *Least-Squares Solution of Linear Inequalities.* Technical Report. DTIC Document.

Nicholas J. Higham. 1991. Algorithm 694: A collection of test matrices in MATLAB. *ACM Transactions on Mathematical Software (TOMS)* 17, 3 (1991), 289–305.

Nicholas J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms*. SIAM.

James K. Ho. 1975. Optimal design of multi-stage structures: A nested decomposition approach. *Computers & Structures* 5, 4 (1975), 249–255.

James K. Ho and Etienne Loute. 1981. A set of staircase linear programming test problems. *Mathematical Programming* 20, 1 (1981), 245–250. DOI:http://dx.doi.org/10.1007/BF01589349

He Huang, John M. Dennis, Liqiang Wang, and Po Chen. 2013. A scalable parallel LSQR algorithm for solving large-scale linear system for tomographic problems: A case study in seismic tomography. *Procedia Computer Science* 18 (2013), 581–590.

George Karypis and Vipin Kumar. 2009. Metis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/~metis.

Charles L. Lawson and Richard J. Hanson. 1987. *Solving Least Squares Problems (Classics in Applied Mathematics)*. Society for Industrial Mathematics.

MATLAB. 2015. *version 8.6.0 (R2015b)*. The MathWorks Inc., Natick, Massachusetts.

Kenji Matsuura and Y. Okabe. 1995. Selective minimum-norm solution of the biomagnetic inverse problem. *IEEE Transactions on Biomedical Engineering* 42, 6 (June 1995), 608–615. DOI:http://dx.doi.org/10.1109/10.387200

Mohammad Javad Peyrovian and Alexander A. Sawchuk. 1978. Image restoration by spline functions. *Applied Optics* 17, 4 (Feb 1978), 660–666. DOI:http://dx.doi.org/10.1364/AO.17.000660

Mustafa Ç. Pinar. 1998. Newton's method for linear inequality systems. *European Journal of Operational Research* 107, 3 (1998), 710–719.

James Reinders. 2007. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly Media, Inc.

Ahmed H. Sameh and Vivek Sarin. 2002. Parallel algorithms for indefinite linear systems. *Parallel Computing* 28, 2 (2002), 285–299.

Michael A. Saunders. 1972. Large-scale linear programming using the cholesky factorization. (1972).

Mrinal K. Sen and Paul L. Stoffa. 2013. *Global Optimization Methods in Geophysical Inversion*. Cambridge University Press.

Tayfun E. Tezduyar and Ahmed Sameh. 2006. Parallel finite element computations in fluid mechanics. *Computer Methods in Applied Mechanics and Engineering* 195, 13–16 (2006), 1872–1884.

Jia-Zhu Wang, Samuel J. Williamson, and Lloyd Kaufman. 1992. Magnetic source images determined by a lead-field analysis: The unique minimum-norm least-squares estimation. *IEEE Transactions on Biomedical Engineering* 39, 7 (1992), 665–675.

Michael S. Zhdanov. 2002. *Geophysical Inverse Theory and Regularization Problems*. Vol. 36. Elsevier.

Received April 2015; revised August 2016; accepted October 2016