Communication-Optimal Parallel and Sequential Eigenvalue/SVD Algorithms

Grey Ballard
Math 221 Numerical Linear Algebra

December 8, 2009

1 Introduction

Algorithms have two costs: arithmetic and communication, by which we mean either moving data between levels of a memory hierarchy (in the sequential case) or over a network connecting processors (in the parallel case). The simplest metric of communication is to count the total number of words moved (also called the bandwidth cost). On current hardware the cost of moving a single word already greatly exceeds the cost of one arithmetic operation, and technology trends indicate that this gap is growing exponentially over time. So it is of interest to devise algorithms that minimize communication, sometimes even at the price of doing more arithmetic.

In this paper we present sequential and parallel algorithms for finding eigendecompositions and SVDs of dense matrices, that do \( O(n^3) \) arithmetic operations, are numerically stable, and do asymptotically less communication than previous such algorithms.

In fact, these algorithms attain known communication lower bounds that apply to many \( O(n^3) \) algorithms in dense linear algebra: In the sequential case, when the \( n \)-by-\( n \) input matrix does not fit in fast memory of size \( M \), the number of words moved between fast (small) and slow (large) memory is \( \Omega(n^3/\sqrt{M}) \).

In the case of \( P \) parallel processors, where each processor has room in memory for \( 1/P \)-th of the input matrix, the number of words moved between one processor and the others is \( \Omega(n^2/\sqrt{P}) \). These lower bounds were originally proven for sequential [HK81] and parallel [ITT04] matrix multiplication, and extended to many other linear algebra algorithms in [BDHS09], including the first phase of conventional eigenvalue/SVD algorithms: reduction to Hessenberg, tridiagonal and bidiagonal forms.

Our algorithms, however, do not rely on reduction to these condensed forms; instead they rely on explicit QR factorization (which is also subject to these lower bounds). This raises the question as to whether there is a communication lower bound independent of algorithm for solving the eigenproblem. Ongoing work includes providing a partial answer by a lower bound for computing the Schur form of a particular block matrix by any algorithm.

We note that there are analogous lower bounds not just for the total number of words moved, but for the number of messages containing these words (also called the latency cost).
Our algorithms also attain these bounds. These algorithms are based on divide-and-conquer algorithms originally formulated in [Mal89], modified to be numerically stable and otherwise more practical in [BDG97], and modified to be randomized in [DDH07], in order to reduce the arithmetic complexity to that of fast matrix multiplication (e.g. $O(n^{2.81})$ using Strassen’s matrix multiplication). Here we begin with the algorithms in [DDH07], and take advantage of the reduction to matrix multiplication and dense QR decomposition, for which an optimal communication-complexity algorithm was given in [DGHL08], to analyze their communication complexity when doing $O(n^2)$ arithmetic operations.

We also make several contributions to the numerical analysis of the methods themselves. First, they depend on a rank-revealing generalized QR decomposition of 2 matrices $A^{-1}B$, which is done with pivoting in [BDG97] and by randomization in [DDH07]. Ongoing work will show in fact that the randomization technique is more reliable because it is provably rank-revealing for the product $A^{-1}B$, whereas in the earlier pivoting technique the pivot order depends only on $B$. More generally, it is possible to compute a randomized rank revealing QR decomposition of an arbitrary product of matrices (and/or inverses).

Second, we elaborate the divide-and-conquer strategy in more detail than previous publications, which differs significantly in the nonsymmetric and symmetric/SVD cases. The symmetric/SVD case returns a list of eigenvalues/eigenvectors (singular values/vectors) as usual. But in the nonsymmetric case, the ultimate output may not be a matrix in Schur form, but rather block Schur form along with a bound (most simply a convex hull) on the $\epsilon$-pseudospectrum of each diagonal block, along with an indication that further refinement of the pseudospectrum is impossible without higher precision. For example, a single Jordan block would be represented (roughly) by a circle centered at the eigenvalue. A conventional algorithm would of course return $n$ eigenvalues evenly spaced along the circular border of the pseudo-spectrum, with no indication (without further computation) that this is the case. We argue that the information returned by divide-and-conquer may sometimes provide more information than the conventional method.

2 Divide and Conquer Algorithm

In this section we describe one divide and conquer step of the algorithm. The goal of the divide and conquer step is to divide the spectrum along a curve in the complex plane and orthogonally transform the matrix to block triangular in order to compute the eigenvalues of the diagonal blocks as subproblems. This step consists of an iterative process that seeks to separate two subsets of eigenvalues followed by a rank-revealing process which extracts the invariant subspace associated with one of the subsets. Algorithm 1 gives the high level algorithm in the case of the ordinary nonsymmetric eigenproblem where the eigenvalues lying inside the unit circle are separated from those lying outside. An analogous algorithm can be used for generalized eigenproblem.
Algorithm 1 Splitting the spectrum of a matrix $A$ along unit circle

1: Implicit Repeated Squaring of $A^{-1}$
2: Compute invariant subspace of $\left(I + (A^{-1})^{2^k}\right)^{-1}$
3: Apply orthogonal transformation to $A$ so that $Q^TAQ = \begin{bmatrix} A_{11} & A_{21} \\ \varepsilon & A_{22} \end{bmatrix}$

2.1 Möbius Transformations

In order to split the spectrum along any line or circle, we can use Möbius transformations which have the form $f(z) = (\alpha z + \beta)^{-1}(\gamma z + \delta)$, for complex constants $\alpha, \beta, \gamma, \delta$. These transformations can map the unit circle to any line or circle in the complex plane. From the spectral mapping theorem, applying the divide and conquer step to $f(A)$ localizes the eigenvalues subject to the curve to which the unit circle is mapped. That is, we implicitly repeatedly square (implicitly) $(\alpha A + \beta)^{-1}(\gamma A + \delta)$ at the first step of Algorithm 1, and we compute the invariant subspace of $\left(I + ((\alpha A + \beta)^{-1}(\gamma A + \delta))^{2^k}\right)^{-1}$ at the second step.

The constants $\alpha, \beta, \gamma, \delta$ must be chosen at each step, and we will discuss how to choose them in Section 3.

2.2 Implicit Repeated Squaring

We want to perform repeated squaring of a product of two matrices, where the first matrix is inverted. So that we do not have to compute the explicit inverse, we use an iterative process involving the QR decomposition; the process is given in Algorithm 2.

Algorithm 2 Implicit Repeated Squaring of $C^{-1}D$

Require: $C_0 = C$, $D_0 = D$

1: repeat
2: $\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \cdot \begin{bmatrix} R_j \\ 0 \end{bmatrix} = qr\left(\begin{bmatrix} D_j \\ -C_j \end{bmatrix}\right)$
3: $C_{j+1} = Q_{12}^* \cdot C_j$
4: $D_{j+1} = Q_{22}^* \cdot D_j$
5: until $R_j$ converges

Ensure: $C_j$, $D_j$

From line 2, we have $Q_{12}^*D_j - Q_{22}^*C_j = 0$, which implies $Q_{12}^*Q_{22}^* = D_jC_j^{-1}$. From lines 3 and 4 we have $C_{j+1}^{-1}D_{j+1} = (C_j^{-1}Q_{12})(Q_{22}^*D_j)$. Thus, for each $j$, Algorithm 2 yields

$C_j^{-1}D_j = (C_{j-1}^{-1}D_{j-1})^2 = (C^{-1}D)^{2^j}$.

2.3 Generalized Randomized Rank-Revealing Decomposition

Because we seek the invariant subspace of matrices of the form $C^{-1}D$, we must extend the randomized rank-revealing decomposition of [DDH07] to products of (inverted) matrices.
Algorithm 3 incorporates the original RURV decomposition of the matrix $D$ in lines 1 and 2, but ultimately produces matrices the comprise a rank-revealing decomposition of $C^{-1}D$. It yields

$$C^{-1}D = U^* (R_1^{-1} R_2) V.$$  

**Algorithm 3** Generalized RURV of $C^{-1}D$

**Require:** $C$, $D$

1: generate random matrix $B$
2: $V \cdot R = \text{qr}(B)$
3: $Q \cdot R_2 = \text{qr}(D \cdot V^*)$
4: $R_1 \cdot U = \text{rq}(Q^* \cdot C)$

**Ensure:** $U$, ($R_1$, $R_2$, $V$)

Assuming there is a gap in the singular values of $C^{-1}D$, the decomposition given above is rank-revealing with high probability. For the purposes of the divide and conquer algorithm, we expect a gap to be produced by the repeated squaring iteration, and we are only interested in using the orthogonal matrix $U$. That is, we do not need to compute $R_1^{-1}$ in order to determine the upper triangular factor in the decomposition of $C^{-1}D$.

### 3 Randomized Bisection

Using the divide and conquer algorithm of Section 2, we hope to generate two smaller subproblems. In this section we describe how to solve the subproblems recursively. As mentioned earlier, we can use Möbius transformations to split along any line or circle in the complex plane, and so we present a systematic approach of randomized bisection in order to choose the constants $\alpha, \beta, \gamma, \delta$ to ensure progress with high probability.

In the case of the symmetric eigenproblem and SVD, all the eigenvalues are real and the analysis is simpler. The splitting technique is to find a bounding interval of the eigenvalues (using Gershgorin’s theorem, for example), set a range about the midpoint of the interval, and then pick a point at random inside the range. We choose the constants of the transformation such that the splitting curve is the line perpendicular to the real axis through the chose point. Figure 1 shows an example of this process.

Choosing a range about the midpoint yields the advantages of bisection (eigenvalues can be localized in a logarithmic number of steps), and using bisection ensures the progress will be made with high probability (choosing a point that is too close to an eigenvalue will cause convergence of the iterative repeated squaring to fail).

In the case of the nonsymmetric (generalized) eigenproblem, the analysis is more complicated. If the matrix is normal (so that the $\varepsilon$-pseudospectrum consists of $\varepsilon$ balls around eigenvalues), we can use a similar approach of randomized bisection and make progress with high probability. In this case, we find a bounding circle, set a range (another circle) around the center of the bounding circle, and then make two random choices. We first draw a line through the center of the circle at a random angle (in $[0, 2\pi]$). Then we randomly pick a
point on that line that falls inside the range, and our splitting curve is the line perpendicular to the first line that intersects the randomly chosen point. Figure 2 shows an example of this process.

Ongoing work will show that in these cases, the random choices will allow progress with high probability. In the case of a non-normal matrix, the $\varepsilon$-pseudospectrum may have arbitrary shape. In this case, we have the ability to compute a convex region of the complex plane (formed with circles and lines) which encloses a region of the pseudospectrum, but we do not currently know of a systematic scheme to determine such an enclosure.

4 Numerical Experiments

In this section we provide a few numerical experiments for one step of divide and conquer. We consider two convergence criteria, as shown in Algorithms 4 and 5. The convergence of the upper triangular matrix $R_j$ is relatively cheap to compute, only requiring the extra
computation of a matrix subtraction and norms while testing the convergence of the south-west block of the transformed matrix \( (E_{21}) \) requires a subroutine call to GRURV and two matrix multiplications. However, the second criterion gives the backward error of applying the transformation and must be verified before solving the subproblems. Since this is the more important criterion, computing it and finding it sufficiently small could possibly save some iterations.

Below are six numerical experiments, each with a plot of the eigenvalues in the complex plane on the left and a plot of the convergence criteria at each iteration on the right (note the y-axis is a log scale). Figure 3 shows a normal matrix with randomly generated eigenvalues, and we see that the convergence is relatively fast (about 10 iterations) and that both convergence criteria drop below a tolerance of \( O(n\varepsilon) \) at about the same time. This example is an “easy” test and we expect convergence with high probability (similar to the probability that for a given normal matrix, a randomized bisection will yield convergence).

Figures 4, 5, and 6 show examples of matrices which are more difficult for the algorithm. In each case, half of the eigenvalues lie just outside the unit circle (at a distance of \( 10^{-5} \) away). In Figure 4, the other eigenvalues are farther away from the unit circle, and we see that while the R convergence is slow, the E convergence criterion is \( O(n\varepsilon) \) after 6 iterations. In this case, it would pay off to measure E convergence and save the work done in the extra iterations. In Figures 5 and 6, the eigenvalues inside the unit circle are pushed out to lie just inside the unit circle. In this way, the unit circle is splitting pairs of eigenvalues which have gaps of \( 2 \cdot 10^{-5} \). As evidenced by the convergence plot in Figure 5, this is a tougher problem for the algorithm, and the measure of the backward error (E convergence criterion) might be too large for the computation. The fix in this case is to choose a different splitting curve (the unit circle is obviously the worst possible splitting curve for these eigenvalues), and we see in Figure 6 convergence after a few iterations.

In Figures 7 and 8, we test the algorithm on a non-normal matrix. Half the eigenvalues form a Jordan block at 1.3 and the other half are clustered near 0. Because the \( \varepsilon \)-pseudospectrum is larger and lies near the unit circle, the splitting problem becomes more ill-conditioned and the backward error suffers as a result. We are continuing tests of such matrices, but one interesting idea of improving the backward error is to restart the iteration. After the matrix \( R_j \) converges (but \( E_{21} \) is not small enough), if we restart the iteration and set \( A_j = A_{\text{new}} \) and \( B_j = I \) in Algorithm 5, we can achieve better results. The green curve in Figure 8 shows the E convergence measure where the algorithm is restarted after 10 iterations. We see that the error jumps back up to \( O(1) \) but quickly converges a second time, and this time it converges to \( O(n\varepsilon) \). This restart is a natural practice in iterative methods, but we still need to understand the mathematical reasons for this behavior. If this mechanism works in general, it can ensure progress in more situations and therefore with higher probability.
Algorithm 4 \( R \) convergence

1. \( A_0 = A, \ B_0 = I \)
2. repeat
3. \[
\begin{bmatrix}
Q_{11} & Q_{12} \\
Q_{21} & Q_{22}
\end{bmatrix}
\begin{bmatrix}
R_j \\
0
\end{bmatrix} = \text{qr}
\begin{bmatrix}
B_j \\
-A_j
\end{bmatrix}
\]
4. \( A_{j+1} = Q_{12}^* \cdot A_j \)
5. \( B_{j+1} = Q_{22}^* \cdot B_j \)
6. until \( \frac{\|R_j - R_{j-1}\|}{\|R_{j-1}\|} \) is small
7. \( U = \text{GRURV}(A_j + B_j, A_j) \)
8. \( A_{\text{new}} = U \cdot A \cdot U^* = 
\begin{bmatrix}
A_{11} & A_{12} \\
E_{21} & A_{22}
\end{bmatrix}
\)

Algorithm 5 \( E \) convergence

1. \( A_0 = A, \ B_0 = I \)
2. repeat
3. \[
\begin{bmatrix}
Q_{11} & Q_{12} \\
Q_{21} & Q_{22}
\end{bmatrix}
\begin{bmatrix}
R_j \\
0
\end{bmatrix} = \text{qr}
\begin{bmatrix}
B_j \\
-A_j
\end{bmatrix}
\]
4. \( A_{j+1} = Q_{12}^* \cdot A_j \)
5. \( B_{j+1} = Q_{22}^* \cdot B_j \)
6. \( U = \text{GRURV}(A_j + B_j, A_j) \)
7. \( A_{\text{new}} = U \cdot A \cdot U^* = 
\begin{bmatrix}
A_{11} & A_{12} \\
E_{21} & A_{22}
\end{bmatrix}
\)
8. until \( \frac{\|E_{21}\|}{\|A\|} \) is small

Figure 3: Normal matrix with random eigenvalues
Figure 4: Normal matrix with half the eigenvalues just outside unit circle (distance $10^{-5}$) and half the eigenvalues clustered around 0

Figure 5: Normal matrix with half the eigenvalues just outside unit circle and half the eigenvalues just inside the unit circle (distance $10^{-5}$)

Figure 6: Normal matrix with half the eigenvalues just outside unit circle and half the eigenvalues just inside the unit circle (distance $10^{-5}$); splitting curve is the imaginary axis
Figure 7: Non-normal matrix with half the eigenvalues forming a Jordan block at 1.3 and the other half clustered around 0.

Figure 8: Non-normal matrix with half the eigenvalues forming a Jordan block at 1.3 and the other half clustered around 0; green curve shows convergence restarting after 10 iterations (setting $A_j = A_{\text{new}}$, $B_j = I$).
References


