Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2013

Parallelizing the conjugate gradient algorithm for multilevel Toeplitz systems☆

Jie Chen[a,*], Tom L. H. Li[b]

[a]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60517, USA*
[b]*Department of Mathematics and Computer Science, University of Missouri–St. Louis, St. Louis, MO 63121, USA*

Abstract

Multilevel Toeplitz linear systems appear in a wide range of scientific and engineering applications. While several fast direct solvers exist for the basic 1-level Toeplitz matrices, in the multilevel case an iterative solver provides the most general and practical solution. This paper proposes several parallelization techniques that enable an efficient implementation of the conjugate gradient algorithm for solving multilevel Toeplitz systems on distributed-memory machines. The two major differences between this implementation and that for a general sparse linear solver are (1) a communication-efficient approach to handle data expansion and truncation and data transpose simultaneously; (2) the interleaving of matrix-vector multiplications and vector inner product calculations to reduce synchronization cost and latency. Similar ideas can be applied to the implementation of other iterative methods for Toeplitz systems that are not necessarily symmetric positive definite. Scaling results are shown to demonstrate the usefulness of the proposed techniques.

Keywords: Toeplitz; conjugate gradient; FFT; all-reduction

1. Introduction

Many scientific and engineering applications give rise to (multilevel) Toeplitz linear systems. Examples include solving partial differential equations, integral equations, digital signal processing, image processing, optimal control, and stationary time series. A Toeplitz matrix has constant diagonals. This special structure enables the development of algorithms that run faster than $O(n^3)$, which is the cost of solving a general, unstructured dense linear system using a direct method. An extensive literature has been devoted to the solution of Toeplitz systems (with parallel implementation), including fast algorithms (such as Levinson-Durbin [1, 2, 3] and Bareiss [4, 5]), "superfast" algorithms using divide-and-conquer strategies (see, e.g., [6, 7, 8]), iterative algorithms based on Newton iterations [9], and algorithms for banded Toeplitz systems (see, e.g., [10, 11]). Some of the algorithms can be generalized for block-Toeplitz or Toeplitz-block systems [12].

A multilevel Toeplitz matrix is defined recursively with respect to the number of levels (sometimes the term "multilevel" is dropped for convenience if the distinction with 1-level is unimportant). In the simplest case, a 2-level Toeplitz matrix is a block-Toeplitz matrix where each block itself is Toeplitz. A d-level Toeplitz matrix

☆This work was supported by the U.S. Department of Energy under Contract DE-AC02-06CH11357. We gratefully acknowledge use of the Fusion cluster in the Laboratory Computing Resource Center at Argonne National Laboratory.

*Corresponding author.
E-mail address: jiechen@mcs.anl.gov (Jie Chen), ll9n8@mail.umsl.edu (Tom Li).

usually comes from a problem with a d-dimensional regular grid structure. Compared with the abundance of algorithms for 1-level Toeplitz systems, algorithms for multilevel Toeplitz systems are rare. One of the reasons is that extending the above algorithms to fully utilize the recursive Toeplitz structure is not straightforward.

Iterative methods (Krylov subspace methods) provide a flexible set of algorithms for solving general linear systems. For symmetric positive definite Toeplitz systems, the conjugate gradient (CG) algorithm with circulant, banded, or similar preconditioners was primarily studied (see, e.g., [13, 14, 15, 16]). Other Krylov algorithms are also applicable; for example, MINRES and GMRES can be used to solve indefinite and unsymmetric Toeplitz systems, respectively (see [17] for a comprehensive treatment of iterative methods). Two crucial computational concerns in applying an iterative method are the efficient multiplication of the matrix with a vector and the efficient construction and application of a preconditioner. For multilevel Toeplitz matrices, the matrix-vector multiplication can be carried out by using fast Fourier transforms (FFTs), as can the application of the preconditioner if it is circulant. This technique lays down the foundation for using iterative methods for solving systems with an arbitrary number of Toeplitz levels.

This paper discusses the parallelization of CG for multilevel Toeplitz systems on distributed-memory machines. The matrix-vector multiplication in this case differs significantly from that in the sparse case. A sparse matrix often arises from discretizations (for example, of differential operators). By domain decomposition, the communication in the multiplication is local. On the other hand, the FFT for multiplying circulant or Teoplitz matrices requires global communications (all-to-all type) because data transpose is needed. What complicates this multiplication is that the Toeplitz matrix and the vector must be expanded in size (called embedding) before the multiplication, and the results must be truncated to yield a correct-sized vector after the multiplication. If not properly implemented, the embedding and truncation incur extra communications. We propose interleaving the embedding and truncation with each substep of the FFT calculation, so that the former can take advantage of the communications in the latter in order to save the extra cost.

The second improvement of the proposed implementation is the handling of vector inner product and norm calculations (to facilitate presentation, throughout this paper we use "inner product" to mean either the inner product or the norm). Inner product calculations are a common bottleneck in parallel iterative solvers because they require global synchronizations. Especially for sparse solvers, synchronizations can constitute over 50% of the overall run time when the CPU cores grow to over thousands. A focus of modern sparse solver design is to modify the numerical algorithm in order to reduce the number of inner product calculations (see, e.g., [18, 19, 20, 21, 22]). In our case, it is also beneficial to consider how to reduce the synchronizations. One will see that with simple mathematical derivations the inner products need not be computed *after* the matrix-vector products, even though in the original CG algorithm it appears so. The inner products can be equivalently expressed by using intermediate results in the matrix-vector multiplication. Therefore, we can utilize the all-to-all communications required in the multiplications to simultaneously compute the inner products, thus eliminating the synchronizations and reducing latency. It is crucial that this rearrangement of the calculation is numerically stable, as demonstrated by experimental results.

2. Preliminaries

2.1. Circulant Matrices, Toeplitz Matrices, and Matrix-Vector Multiplications

A circulant matrix C and a Toeplitz matrix T, of order n, are defined in the following forms, respectively:

$$C = \begin{bmatrix} c_0 & c_{n-1} & c_{n-2} & \cdots & c_1 \\ c_1 & c_0 & c_{n-1} & \ddots & \vdots \\ c_2 & c_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & c_{n-1} \\ c_{n-1} & \cdots & \cdots & c_1 & c_0 \end{bmatrix}, \qquad T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \cdots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \ddots & \vdots \\ t_2 & t_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \cdots & \cdots & t_1 & t_0 \end{bmatrix}. \tag{1}$$

We will use a subscript n to emphasize the order when necessary. In (1), if each c_i itself is a circulant block, then the resulting matrix is 2-level circulant. In this case, if i ranges from 0 to $n_0 - 1$ and each c_i has size $n_1 \times n_1$,

we say that C is of order (n_0, n_1). In this manner, a d-level circulant matrix of order $(n_0, n_1, \cdots, n_{d-1})$ is defined recursively based on a $(d-1)$-level circulant matrix of order $(n_0, n_1, \cdots, n_{d-2})$. A multilevel Toeplitz matrix is similarly defined.

Since this paper addresses real symmetric matrices, the multilevel C and T can be represented solely by their first columns. Such columns can be reshaped to a d-dimensional data tensor because of the recursive nature of the definition of multilevels. Thus, we use the notation c and t, both having size $n_0 \times n_1 \times \cdots \times n_{d-1}$, to mean a data representation of C and T, of order $(n_0, n_1, \cdots, n_{d-1})$, respectively.

For notational convenience, we always let $n = n_0 \cdots n_{d-1}$. The eigenvalues of C can be stored in a data tensor λ, which is obtained by a d-dimensional FFT of c followed by a multiplication with $\sqrt{n}$. We sometimes drop the term "d-dimensional" in front of FFT if the context is clear. Correspondingly, let the tensor y be the data representation of a vector $\boldsymbol{y}$. Then, the matrix-vector product $\boldsymbol{v} = C\boldsymbol{y}$ can be represented as a tensor v, which is obtained by elementwise product between λ and the FFT of y, followed by an inverse FFT of the product.

A d-level Toeplitz matrix T can be embedded into a d-level circulant matrix C of twice the size in each level. It suffices to show a 1-level example and a 2-level example. When $d = 1$, the embedding is in the following form:

$$C_{2n} = \begin{bmatrix} T_n & * \\ * & T_n \end{bmatrix}, \tag{2}$$

where the entries of $*$ are used to fulfill that C_{2n} is circulant. In the data representation, if c is the embedding of t, then c has a length $2n$, where the first n entries of c is t, the next entry is arbitrary, and the rest of the $n-1$ entries are the second to the nth entry of t, in the reverse order. Informally, we say that the second half of c is a "flipping" of t. When $d = 2$, c has the form

$$\mathsf{c} = \begin{bmatrix} \mathsf{t} & * \\ * & * \end{bmatrix}, \tag{3}$$

which graphically means that it is obtained by flipping not only the rows, but also the columns, of t. That is, a circulant embedding should expand t along each direction.

The use of an embedding is to multiply a Toeplitz matrix with a vector. When $d = 1$, we make use of the following identity:

$$\begin{bmatrix} T_n\boldsymbol{y} \\ * \end{bmatrix} = \begin{bmatrix} T_n & * \\ * & T_n \end{bmatrix} \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{bmatrix}. \tag{4}$$

Thus, to obtain the matrix-vector product $\boldsymbol{v} = T_n\boldsymbol{y}$, one embeds T_n to C_{2n} and embeds $\boldsymbol{y}$ to $\boldsymbol{y}'$ (by padding zeros instead of flipping), computes the matrix-vector product $\boldsymbol{v}' = C_{2n}\boldsymbol{y}'$, and then rids the second half of $\boldsymbol{v}'$. It is straightforward to generalize this procedure to the multilevel case with the data representations of the matrices and vectors.

2.2. Circulant Preconditioner for Toeplitz Matrices

For solving a linear system with respect to a 1-level Toeplitz matrix T, a matrix M with a circulant structure is proved to be effective [13]. Several choices of such preconditioners exist; in this paper we consider T. Chan's preconditioner [14]. It modifies the spectrum of T to yield clustered eigenvalues, so that an iterative method can converge superlinearly. When generalized to the multilevel case, although the superlinear convergence property is lost [23], the preconditioner still yields sufficiently good performance in practice [13].

Because M is multilevel circulant, let the tensor m be the data representation of M. The construction of m from t is based on the following formula

$$\mathsf{m}_{\cdots,j,\cdots} = \frac{1}{n_i}((n_i - j)\,\mathsf{t}_{\cdots,j,\cdots} + j\,\mathsf{t}_{\cdots,n_i-j,\cdots}), \qquad \text{for all } j = 0, \ldots, n_i - 1 \text{ and } i = 0, \ldots, d-1, \tag{5}$$

where in the subscript notation "$\cdots, j, \cdots$", j is the ith index of the data tensor. Informally, along each dimension, a fiber of m is a weighted averaging of a fiber of t and its flipping, and the weights are defined with respect to the locations of the entries. For more details, see [13].

2.3. Parallel Multidimensional FFT

FFTs are required to multiply a Toeplitz matrix T, and also a circulant preconditioner M, to vectors. We use z to mean a general $n_0 \times \cdots \times n_{d-1}$ data tensor in the complex field. A d-dimensional FFT on z is equivalent to d consecutive 1-dimensional FFTs along each dimension of z.

There exist several parallel implementations of multidimensional FFT, such as FFTW [24] and P3DFFT [25]. The former partitions z along one dimension whereas the latter particularly handles 3-dimensional data by partitioning it along two dimensions, using a 2-dimensional grid of processors. One can generalize the latter approach for d-dimensional data by partitioning it along d' dimensions ($d' < d$). An advantage of using $d' > 1$ for large d is that the number of processors can maximally scale to the product of the size of the corresponding d' dimensions of the data.

3. Parallel Toeplitz Matrix-Vector Multiplication

One difficulty for multiplying a Toeplitz matrix with a vector in parallel is that the embedding incurs interprocessor communications. As an example, consider a 2-level Toeplitz matrix T with the data representation t. Then, the data embedding is in the form

$$\mathsf{c} = \begin{bmatrix} \mathsf{t} & * \\ * & * \end{bmatrix}. \tag{6}$$

Suppose t is partitioned horizontally by using two processors. In order that c is similarly partitioned, one needs to fill the lower part of c. Furthermore, one needs to redistribute the rows of t. Communication is inevitable if one wants to construct c explicitly.

To avoid this communication, one can combine the steps of the embedding and the FFT, since the embedding can take advantage of the transposes in the FFT for data redistribution. The details are best explained by walking through Figure 1(a), which showcases the situation that the data is partitioned along only one dimension. The flowchart mainly demonstrates the steps of the multiplication; however, the first phase of the flow can also be used to compute the eigenvalues of the embedding of the Toeplitz matrix.

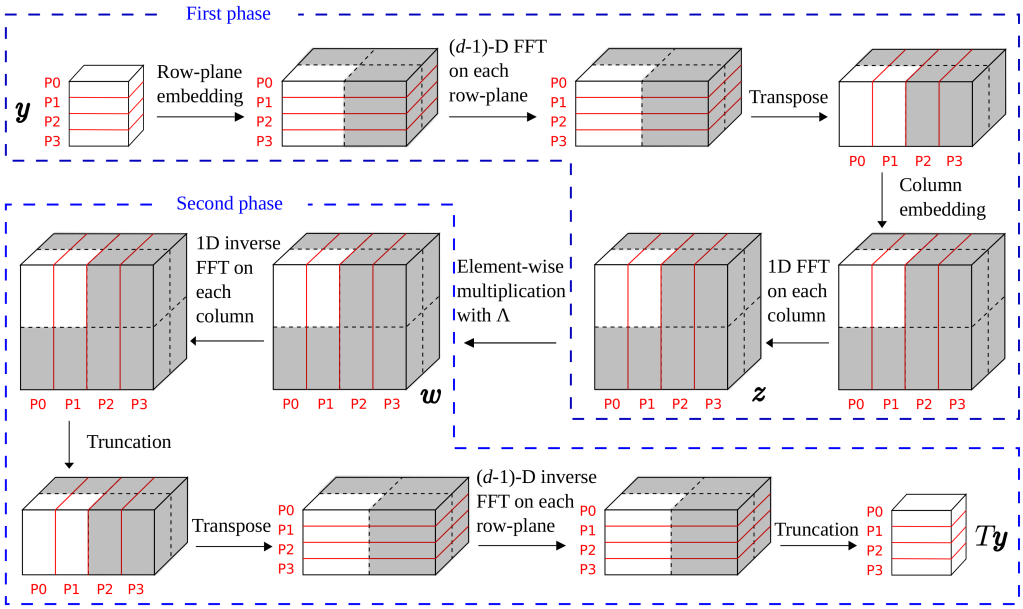
To compute $\boldsymbol{v} = T\boldsymbol{y}$, we need to embed y into a larger data tensor by padding zeros. The first step is to embed along the 2nd to the dth dimensions, shown in the figure as "row-plane embedding." After the embedding, we immediately perform a $(d-1)$-dimensional FFT on these dimensions. Next, a transpose between the 1st dimension and the 2nd to the dth dimensions is carried out, so that data along the 1st dimension become local. Then, we perform a further embedding (zero-padding) along the 1st dimension, followed immediately by a 1-dimensional FFT along this dimension. This in fact completes the d-dimensional FFT on the full embedding of y. The result is denoted by z.

The procedure for obtaining the eigenvalues λ of the circulant embedding of t is similar, except that the "zero-padding" is changed to "flipping." Corresponding to the figure, the first flipping is to embed each row-plane to a larger data tensor such that it represents a $(d-1)$-level circulant matrix, and the second flipping is to embed each column to a longer column such that it represents a 1-level circulant matrix.

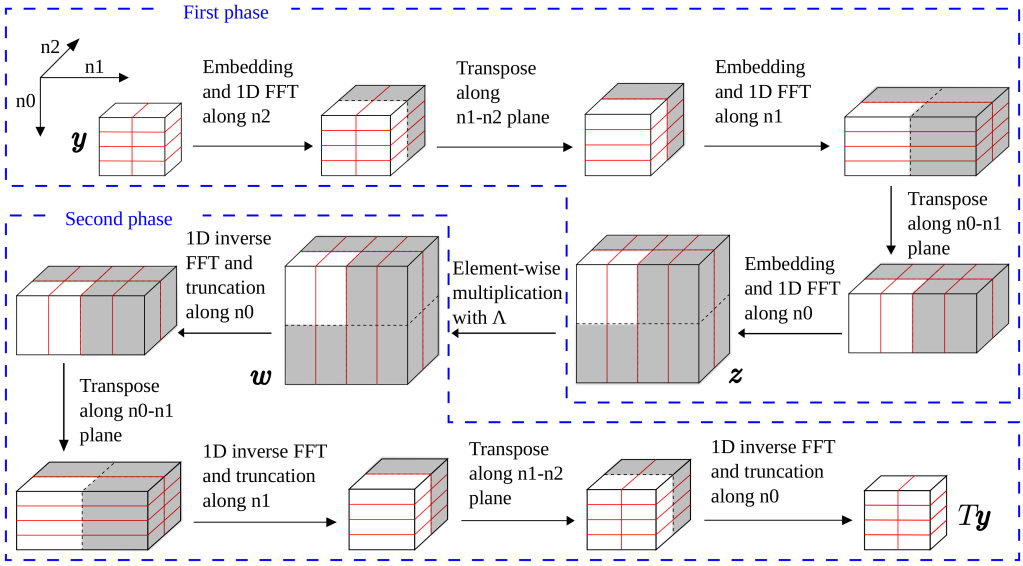
With z and λ ready, an elementwise multiplication is performed, resulting in w. The final step is to perform a d-dimensional inverse FFT and the truncation simultaneously, in order to obtain the final result v. This, in fact, is the reverse procedure of obtaining z, and it corresponds to the second phase of Figure 1(a). We perform a 1-dimensional inverse FFT along the 1st dimension, followed by a truncation that rids the latter half of the data along this dimension. Next, a transpose is performed so that the data is cut along the 1st dimension. Then, a $(d-1)$-dimensional inverse FFT along the 2nd to dth dimensions is carried out, and the data is truncated to half along each of these dimensions. The result is the data v.

Similar ideas apply to the situation where the data is partitioned along more than one dimension. Figure 1(b) shows an example of a 3-dimensional data partitioned along the first two dimensions. The procedure begins with embedding and FFT along the unpartitioned dimension(s), then iteratively performing transposes so that each time the data along a new dimension becomes local; thus, embedding and FFT are done along this dimension. After the iterative loop, both z and λ are ready, so an elementwise multiplication is performed, resulting in w. Then, w goes through the reverse procedure (inverse FFT and truncation), and v is obtained.

The discussions are summarized in Algorithm 1.



(a) 1-dimensional partitioning



(b) 2-dimensional partitioning

Fig. 1. Flowcharts of computing Toeplitz matrix-vector product $\boldsymbol{v} = T\boldsymbol{y}$. This procedure corresponds to the subroutine Toep-Mult($\Lambda, \boldsymbol{y}, \boldsymbol{v}$) in Algorithm 1, where Λ contains the eigenvalues of the embedding of T. The first phase of each chart also shows the steps of performing the subroutine Toep-Embed-EigVal(T, Λ).

4. Eliminating All-Reduction

A common bottleneck in parallel iterative solvers is the inner product calculations, because each calculation requires an all-reduction operation to sum the local inner products held in each processor in order to obtain the global result. The all-reduction incurs a global synchronization, and there are several such synchronizations within each iteration.

The idea of removing these repeating synchronizations is to hide the latency in other global communications

Algorithm 1 Toeplitz matrix times vector and eigenvalues of embedding, parallel version

1: **function** TOEP-MULT($\Lambda, \boldsymbol{y}, \boldsymbol{v}$)
2: Embed y and compute d-dimensional FFT of the embedding simultaneously (corresponding to the first phase of Figure 1(a)/1(b); embedding means "zero-padding"). Let the result be z.
3: Compute w as the elementwise product of λ an z.
4: Compute d-dimensional inverse FFT of w and truncate the FFT result simultaneously (corresponding to the second phase of Figure 1(a)/1(b)). The end result is v.
5: **end function**
6: **function** TOEP-EMBED-EIGVAL(T, Λ)
7: Embed t and compute d-dimensional FFT of the embedding simultaneously (corresponding to the first phase of Figure 1(a)/1(b); embedding means "flipping").
8: Obtain λ by scaling the above result by $\sqrt{n}$.
9: **end function**

(in our case, the all-to-alls for transposing data). Thus, *if the data for all-reduction is available at the time of transpose*, then the all-reduction can be equivalently carried out by first performing a local (partial) sum, then transmitting the partial sum by all-to-all, followed by a summation of the gathered partial sums.

It is, however, not obvious in the standard CG Algorithm why the data for all-reduction is ready when transpose is being carried out. For example, there occurs a Toeplitz matrix-vector multiplication $\boldsymbol{v} = T\boldsymbol{y}$ and a vector inner product $\sigma = (\boldsymbol{y}, \boldsymbol{v})$. It appears that the latter cannot be computed until the former has been calculated.

In order to compute the two terms simultaneously, we derive a mathematically equivalent way to compute σ. Let $\boldsymbol{y}'$ be the embedding of $\boldsymbol{y}$, and let $C = U^H \Lambda U$ be the diagonalization of the embedding C of T. Then computing $\boldsymbol{v}$ follows these steps: 1. $\boldsymbol{z} = U\boldsymbol{y}'$ (first phase of Figure 1(a)/1(b)); 2. $\boldsymbol{w} = \Lambda \boldsymbol{z}$; 3. $\boldsymbol{v}' = U^H \boldsymbol{w}$, and $\boldsymbol{v}$ is the truncation of $\boldsymbol{v}'$ (second phase of Figure 1(a)/1(b)). Thus, the inner product can be equivalently expressed as

$$(\boldsymbol{y}, \boldsymbol{v}) = (\boldsymbol{y}', \boldsymbol{v}') = (\boldsymbol{w}, \boldsymbol{z}). \tag{7}$$

The first equality results from the fact that $\boldsymbol{y}$ is embedded into $\boldsymbol{y}'$ with zeros, and the second equality is because U is unitary. Therefore, σ can be computed as the inner product of $\boldsymbol{w}$ and $\boldsymbol{z}$, which are available before step 3.

Similarly, we consider the circulant multiplication $\boldsymbol{v} = C\boldsymbol{y}$ and the inner product $\sigma = (\boldsymbol{y}, \boldsymbol{v})$. When C is diagonalized as $U^H \Lambda U$, computing $\boldsymbol{v} = C\boldsymbol{y}$ follows these steps: 1. $\boldsymbol{z} = U\boldsymbol{y}$; 2. $\boldsymbol{w} = \Lambda \boldsymbol{z}$; 3. $\boldsymbol{v} = U^H \boldsymbol{w}$. Therefore, the inner product $(\boldsymbol{y}, \boldsymbol{v}) = (\boldsymbol{w}, \boldsymbol{z})$, and this computation can be inserted between steps 2 and 3.

Based on this idea, we introduce two subroutines, TOEP-MULT-AND-DOT-PROD($\Lambda, \boldsymbol{y}, \boldsymbol{v}, \sigma$) and CIRC-MULT-AND-DOT-PROD-AND-CONVG-TEST($\Lambda, \boldsymbol{y}, \boldsymbol{v}, \sigma, tol$). The former subroutine computes the Toeplitz matrix-vector product $\boldsymbol{v} = T\boldsymbol{y}$ and the inner product $\sigma = (\boldsymbol{y}, \boldsymbol{v})$ simultaneously, where Λ contains the eigenvalues of the circulant embedding of T. The latter subroutine computes the circulant matrix-vector product $\boldsymbol{v} = C\boldsymbol{y}$ and the inner product $\sigma = (\boldsymbol{y}, \boldsymbol{v})$ simultaneously, where Λ contains the eigenvalues of C. This subroutine also computes the vector norm $\rho = \|\boldsymbol{y}\|$ and returns earlier when $\rho < tol$ (which indicates convergence of the CG iterations, as will be clear in the next section). See Algorithm 2.

5. Overall Algorithm

Summarizing the preceding discussions, Algorithm 3 is the CG method for solving a linear system of equations

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{8}$$

where A is multilevel Toeplitz, by using a multilevel circulant preconditioner M. The parameters *rtol* and *maxit* mean the relative residual tolerance and maximum number of iterations, respectively.

The major differences between Algorithm 3 and the standard form of CG (see, e.g., [17, p. 263]) are twofold. First, the eigenvalue calculation of the embedding of A and that of M is brought to the front as a preprocessing step, as shown in lines 1 to 4. This step can be separated from the main CG iteration if there are several right-hand sides $\boldsymbol{b}$'s. Second, matrix-vector multiplications are interleaved with inner product calculations to reduce global synchronizations, as shown in lines 6, 9 and 11.

Algorithm 2 Subroutines eliminating all-reduction

function CIRC-MULT-AND-DOT-PROD-AND-CONVG-TEST($\Lambda, \boldsymbol{y}, \boldsymbol{v}, \sigma, tol$)
  Compute d-dimensional FFT of y and $\rho = \|\mathsf{y}\|$ simultaneously. Let the FFT result be z.
  Return with indication of $\rho < tol$ if so.
  Compute w as the elementwise product of λ an z.
  Compute d-dimensional inverse FFT of w, the result being v. Meanwhile, compute $\sigma = (\mathsf{w}, \mathsf{z})$.
end function
function TOEP-MULT-AND-DOT-PROD($\Lambda, \boldsymbol{y}, \boldsymbol{v}, \sigma$)
  Embed y and compute d-dimensional FFT of the embedding simultaneously (corresponding to the first phase of Figure 1(a)/1(b); embedding means "zero-padding"). Let the result be z.
  Compute w as the elementwise product of λ an z.
  Compute d-dimensional inverse FFT of w and truncate the FFT result simultaneously (corresponding to the second phase of Figure 1(a)/1(b)). The end result is v. Meanwhile, compute $\sigma = (\mathsf{w}, \mathsf{z})$.
end function

Algorithm 3 CG for multilevel Toeplitz systems $A\boldsymbol{x} = \boldsymbol{b}$ with initial guess $\boldsymbol{x}_0$

// The following can be separated out for multiple $\boldsymbol{b}$'s
Compute eigenvalues Λ_1 of the multilevel circulant embedding of A by calling TOEP-EMBED-EIGVAL(A, Λ_1)
Construct multilevel circulant preconditioner M
Compute eigenvalues Λ_2 of M
Compute $\boldsymbol{y}_0 = A\boldsymbol{x}_0$ by calling TOEP-MULT($\Lambda_1, \boldsymbol{x}_0, \boldsymbol{y}_0$)
// Work for each $\boldsymbol{b}$
Compute $\gamma = \|\boldsymbol{b}\|$, $tol = \gamma \cdot rtol$, and $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{y}_0$
Compute $\boldsymbol{z}_0 = M^{-1}\boldsymbol{r}_0$ and $\sigma_0 = (\boldsymbol{r}_0, \boldsymbol{z}_0)$ simultaneously by calling CIRC-MULT-AND-DOT-PROD-AND-CONVG-TEST($\Lambda_2^{-1}, \boldsymbol{r}_0, \boldsymbol{z}_0, \sigma_0, tol$). Return if converged.
Let $\boldsymbol{p}_0 = \boldsymbol{z}_0$
for $j = 0, 1, \ldots, maxit$ **do**
  Compute $\boldsymbol{v}_j = A\boldsymbol{p}_j$ and $\tau_j = (\boldsymbol{v}_j, \boldsymbol{p}_j)$ simultaneously by calling TOEP-MULT-AND-DOT-PROD($\Lambda_1, \boldsymbol{p}_j, \boldsymbol{v}_j, \tau_j$)
  Compute $\alpha_j = \sigma_j/\tau_j$, $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j\boldsymbol{p}_j$, and $\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j\boldsymbol{v}_j$
  Compute $\boldsymbol{z}_{j+1} = M^{-1}\boldsymbol{r}_{j+1}$ and $\sigma_{j+1} = (\boldsymbol{r}_{j+1}, \boldsymbol{z}_{j+1})$ simultaneously by calling CIRC-MULT-AND-DOT-PROD-AND-CONVG-TEST($\Lambda_2^{-1}, \boldsymbol{r}_{j+1}, \boldsymbol{z}_{j+1}, \sigma_{j+1}, tol$). Return if converged.
  Compute $\beta_j = \sigma_{j+1}/\sigma_j$ and $\boldsymbol{p}_{j+1} = \boldsymbol{z}_{j+1} + \beta_j\boldsymbol{p}_j$
end for

6. Experimental Results

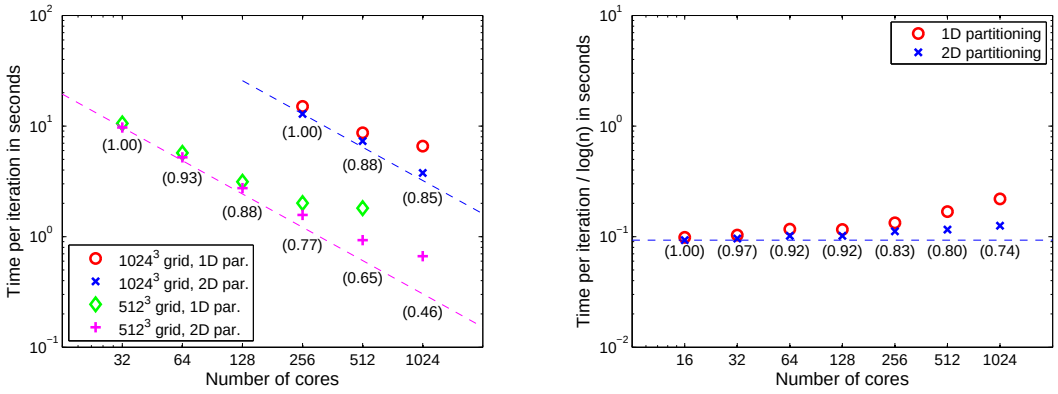
In this section, we show several experiments to demonstrate the effectiveness of the proposed parallelization strategies. The programs were implemented in C, compiled with MVAPICH2 and GCC. The all-to-alls were implemented by directly using MPI_Alltoall(v). The in-processor serial FFTs were called from the FFTW library [24]. The experiments were performed on a cluster with 2,560 computing cores and an InfiniBand QDR network.

The Toeplitz matrix was generated from the Matérn kernel [26, 27, 28], which is positive definite for any number of dimensions, thus CG can be applied with guaranteed convergence. This kernel is widely used in spatial statistics, and the solution of the respective linear system is required in numerous statistical analysis tasks, such as regression and maximum likelihood estimation [26, 27, 28, 29].

The linear systems in the experiments were all 3-level Toeplitz thus the data was 3-dimensional. Therefore, partitioning of the data could be performed along one dimension or two dimensions. We call these 1D and 2D partitionings, respectively. The performances of the solver were significantly different under the two partitioning schemes, as will be shown.

Figure 2 shows the strong and weak scalings (together with parallel efficiency) for one CG iteration. The dashed lines indicate perfect scaling. For each scaling trend in plot (a), the leftmost marker indicates the use of the minimum number of cores such that all the solver data can be fit into the main memory. Because of hardware

limitations, the maximum number of cores used in the experiments was 1,024. We observe the following. First, 2D partitioning offers clear advantages over 1D partitioning. For the former, not only is the running time shorter, but also it scales better. (Moreover, 2D partitioning can utilize more processors to further reduce the running time.) Second, plot (a) shows a satisfactory behavior of the strong scaling in the 2D partitioning case, as seen from the parallel efficiency. In plot (b), the overall trend for 2D partitioning is also favorable.

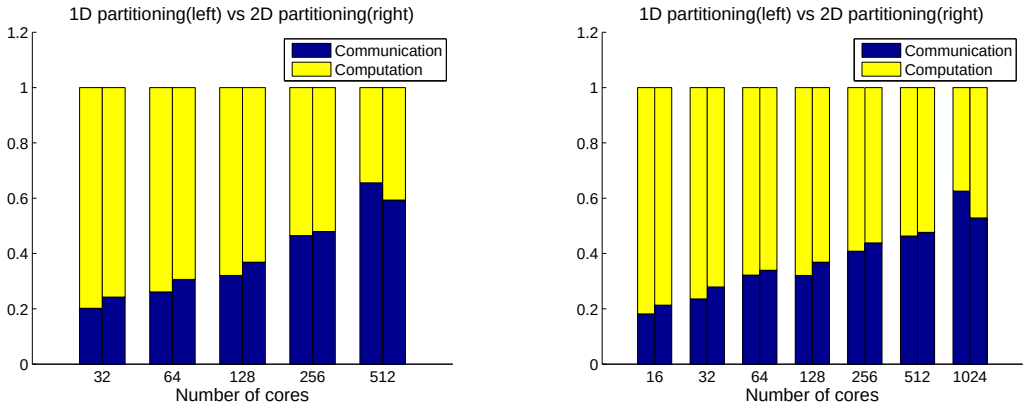


(a) Strong scaling

(b) Weak scaling (2^{20} grid points per core)

Fig. 2. Scalings of running time for each iteration. The numbers in parentheses are the parallel efficiency for 2D partitioning.

The fact that 2D partitioning is superior over 1D partitioning is further demonstrated in Figure 3, which shows the proportions of computation time and communication time. The left plot corresponds to the scenario of a fixed grid size, whereas the right plot corresponds to a fixed grid size per core. One sees that as the number of cores increases, in both partitioning schemes the proportion of communication time increases. However, the increase for 2D partitioning is far slower than that for 1D partitioning. In fact, even though the proportion of communication time in 1D partitioning is smaller at the beginning, it catches up quickly and constitutes a majority part of the overall run time.



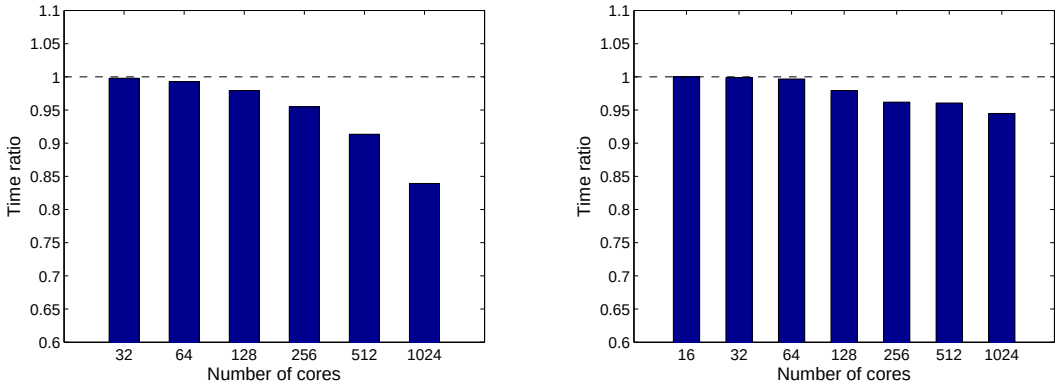
(a) Fixed grid size $512 \times 512 \times 512$

(b) Fixed 2^{20} grid points per core

Fig. 3. Computation and communication time split up.

To demonstrate the effectiveness of the elimination of all-reductions, we show in Figure 4 the ratio of the run time per iteration when the solver is implemented by interleaving the inner product calculations with matrix-vector multiplications (cf. Algorithm 3), over the running time when the inner products are calculated by using all-reduction. Clearly, a ratio less than 1 shows the advantage of the proposed algorithm. Only the results of 2D

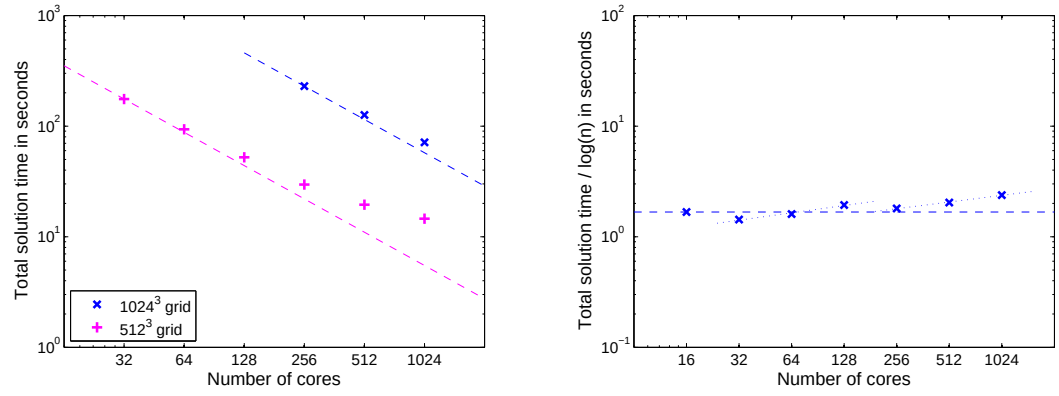
partitioning are shown because we have demonstrated that it is superior over 1D partitioning. Both plots in the figure show that as the number of cores increases, the ratio decreases. For example, in plot (a), the savings by eliminating all-reductions are more than 15% with 1,024 cores. This is expected because the synchronization cost and the variance of time for processors entering the synchronization point is high. We project that the savings will be more significant as the number of cores increases.



(a) Fixed grid size $512 \times 512 \times 512$ (b) Fixed 2^{20} grid points per core

Fig. 4. Time improvement by interleaving inner products with matrix-vector multiplications. Only results of 2D partitioning are shown.

All the above figures were drawn with respect to one iteration. We thus also show the scalings of the overall run time; see Figure 5. The setting of these plots are similar to that of Figure 2, and the results are similar, too. Since the increase of the grid size causes the respective linear system to be increasingly ill-conditioned (even with the use of a multilevel circulant preconditioner), the variation in the iteration counts becomes an undesired factor in the weak scaling test. To improve the solver, we used a filtering technique proposed in [30] to further reduce the condition number. With this technique, the number of iterations varies only slightly. In plot (b), the numbers of iterations for the solver to converge to a relative tolerance of 10^{-6} are 15, 12, 13, 15, 13, 14, 15, respectively. One sees an interesting pattern that for every three problem sizes the numbers of iterations vary roughly in a periodic manner. Then in plot (b) one observes a similar pattern for every three consecutive markers.



(a) Strong scaling (b) Weak scaling (2^{20} grid points per core)

Fig. 5. Scalings of the running time for solving the linear system. Only results of 2D partitioning are shown.

It is worth noting that the mathematically equivalent rearrangement of the inner product calculation maintains the numerical stability of the algorithm according to the recorded number of iterations.

7. Conclusion

Solving a large-scale linear system with respect to a multilevel Toeplitz matrix is required in various science and engineering applications. An iterative Krylov solver provides a principled framework for the solution of such a linear system. We have proposed a parallel implementation of the CG algorithm and shown its effectiveness in an example Toeplitz kernel that is popularly used in spatial statistics. The implementation addresses the reducing of communication cost and latency, by designing the Toeplitz matrix-vector multiplication such that data embedding and truncation are injected into each substep of a multidimensional FFT, and by interleaving the matrix-vector multiplications with inner product calculations to eliminate all-reduction synchronizations. The general idea of the parallel strategies can be used in implementing Krylov solvers other than CG for solving Toeplitz linear systems that are not necessarily symmetric positive definite. The idea may also be useful for other applications (such as electronic structure calculations [31, 32]) where FFT is a major computational component.

References

[1] N. Levinson, The Wiener RMS error criterion in filter design and prediction, J. Math. Phys. 25 (1947) 261–278.
[2] J. Durbin, The fitting of time-series models, Review of the International Statistical Institute 28 (3) (1960) 233–244.
[3] I. Gohberg, I. Koltracht, A. Averbuch, B. Shoham, Timing analysis of a parallel algorithm for Toeplitz matrices on a MIMD parallel machine, in: Proceedings of International Conference on Parallel Processing, 1991.
[4] E. H. Bareiss, Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices, Numer. Math. 13 (1969) 404–424.
[5] R. P. Brent, Parallel algorithms for Toeplitz systems, in: G. H. Golub, P. V. Dooren (Eds.), Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms, Springer-Verlag, 1991.
[6] G. S. Ammar, W. B. Gragg, Superfast solution of real positive definite Toeplitz systems, SIAM J. Matrix Anal. Appl. 9 (1) (1988) 61–76.
[7] M. Stewart, A superfast Toeplitz solver with improved numerical stability, SIAM J. Matrix Anal. Appl. 25 (3) (2003) 669–693.
[8] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, J. Zhu, A superfast algorithm for Toeplitz systems of linear equations, SIAM J. Matrix Anal. Appl. 29 (4) (2007) 1247–1266.
[9] V. Y. Pan, Concurrent iterative algorithm for Toeplitz-like linear systems, IEEE Trans. Parallel Distrib. Syst. 4 (5) (1993) 592–600.
[10] J. Grcar, A. Sameh, On certain parallel Toeplitz linear system solvers, SIAM J. Sci. Stat. Comput. 2 (2) (1981) 238–256.
[11] X.-H. Sun, A scalable parallel algorithm for periodic symmetric Toeplitz tridiagonal systems, Int. J. Comp. Res. 10 (1) (2001) 89–98.
[12] P. Alonso, J. M. Badá, A. M. Vidal, An efficient parallel algorithm to solve block-Toeplitz systems, The Journal of Supercomputing 32 (3) (2005) 251–278.
[13] R. H.-F. Chan, X.-Q. Jin, An Introduction to Iterative Toeplitz Solvers, SIAM, 2007.
[14] T. Chan, An optimal circulant preconditioner for Toeplitz systems, SIAM J. Sci. Stat. Comput. 9 (4) (1988) 766–771.
[15] R. H. Chan, P. T. P. Tang, Fast band-Toeplitz preconditioners for Hermitian Toeplitz systems, SIAM J. Sci. Comput. 15 (1) (1994) 164–171.
[16] D. Bini, F. Benedetto, A new preconditioner for the parallel solution of positive definite Toeplitz systems, in: Proceedings of the second annual ACM Symposium on Parallel Algorithms and Architectures, 1990.
[17] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, SIAM, 2003.
[18] M. Mohiyuddin, M. Hoemmen, J. Demmel, K. Yelick, Minimizing communication in sparse matrix solvers, in: Proceedings of the Conference on High Performance Computing, Networking, Storage and Analysis, 2009.
[19] J. Rosendale, Minimizing inner product data dependencies in conjugate gradient iteration, in: Proceedings of the ICPP, 1983.
[20] A. T. Chronopoulos, C. W. Gear, s-step iterative methods for symmetric linear systems, J. Comput. Appl. Math. 25 (2) (1989) 153–168.
[21] E. de Sturler, H. van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, Appl. Numeri. Math. 18 (4) (1995) 441–459.
[22] L. T. Yang, R. P. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Proceedings of International Conference on Algorithms and Architectures for Parallel Processing, 2002.
[23] S. S. Capizzano, Matrix algebra preconditioners for multilevel Toeplitz matrices are not superlinear, Linear Algebra Appl. 343–344 (1) (2002) 303–319.
[24] M. Frigo, S. G. Johnson, The design and implementation of FFTW3, Proceedings of the IEEE 93 (2) (2005) 216–231.
[25] P3DFFT. `http://code.google.com/p/p3dfft/`.
[26] J.-P. Chilès, P. Delfiner, Geostatistics: Modeling Spatial Uncertainty, Wiley-Interscience, 1999.
[27] M. Stein, Interpolation of Spatial Data: Some Theory for Kriging, Springer-Verlag, 1999.
[28] H. Wendland, Scattered Data Approximation, Cambridge University Press, 2005.
[29] M. Anitescu, J. Chen, L. Wang, A matrix-free approach for solving the parametric gaussian process maximum likelihood problem, SIAM J. Sci. Comput. 34 (1) (2012) A240–A262.
[30] M. L. Stein, J. Chen, M. Anitescu, Difference filter preconditioning for large covariance matrices, SIAM J. Matrix Anal. Appl. 33 (1) (2012) 52–72.
[31] A. Canning, Scalable parallel 3d ffts for electronic structure codes, in: VECPAR, 2008.
[32] A. Canning, J. Shalf, N. Wright, S. Anderson, M. Gajbe, A hybrid MPI/OpenMP 3d FFT for plane wave first-principles materials science codes, in: Proceedings of CSC12 Conference, 2012.