A Parallel Prefix Algorithm for Almost Toeplitz Tridiagonal Systems*

Xian-He Sun
Dept. of Computer Science
Louisiana State University
Baton Rouge, LA 70803-4020

Ronald D. Joslin
Mail Stop 170
NASA Langley Research Center
Hampton, VA 23681-0001

Abstract

A compact scheme is a discretization scheme that is advantageous in obtaining highly accurate solutions. However, the resulting systems from compact schemes are tridiagonal systems that are difficult to solve efficiently on parallel computers. Considering the almost symmetric Toeplitz structure, a parallel algorithm, *simple parallel prefix* (SPP), is proposed. The SPP algorithm requires less memory than the conventional LU decomposition and is efficient on parallel machines. It consists of a prefix communication pattern and AXPY operations. Both the computation and the communication can be truncated without degrading the accuracy when the system is diagonally dominant. A formal accuracy study has been conducted to provide a simple truncation formula. Experimental results have been measured on a MasPar MP-1 SIMD machine and on a Cray 2 vector machine. Experimental results show that the simple parallel prefix algorithm is a good algorithm for symmetric, almost symmetric Toeplitz tridiagonal systems and for the compact scheme on high-performance computers.

*This research was supported in part by the National Aeronautics and Space Administration under NASA contract NAS1-19480 and NAS1-1672.

1 Introduction

Recent technological advances have made it possible to build computers that contain thousands of processors and can obtain gigaflops (10^9 floating-point operations per second) on real applications. Emerging parallel computers are designed to solve large problems and achieve better accuracy than could previously be obtained [22]. Parallel computers demand new models, new discretization methods, and new algorithms to explore the potential of high-performance computing.

Conventionally, partial differential equations (PDEs) are discretized by finite-difference or finite-element methods, and are solved by Gauss-Seidel, conjugate-gradient, or successive overrelaxation (SOR) methods. A new discretization method, the *compact finite-difference scheme* (*or compact-difference scheme, compact scheme*) was proposed by Kreiss and Oliger [12] and was later improved upon by Lele [15]. Compared with the traditional finite-difference scheme, the compact finite-difference scheme achieves higher accuracy with smaller difference stencils and leads to more accurate approximations because of the smaller coefficients on the truncation error. With these advantages, the compact scheme has quickly gained popularity. In practice, the resulting discretized system of the compact schemes are tridiagonal systems that can be solved efficiently on sequential machines. However, tridiagonal systems are difficult to solve efficiently on parallel computers. For example, to study the physics of compressible homogeneous turbulence, the CDNS (compressible direct simulation of Navier-Stokes) code, based on a sixth-order compact scheme and a third-order time discretization, was implemented on the Intel Delta [7]. After carefully choosing an existing tridiagonal solver, mapping the algorithm to the architecture, and overlapping communication with computation, the communication overhead consumed about 30 percent of the total execution time. Clearly, more efficient algorithms are needed to explore the potential of compact schemes on parallel computers.

In recent years, intensive research has been done to develop efficient tridiagonal solvers on parallel computers. A good survey can be found in references [17], [9], and [13]. Of the known tridiagonal solvers, both the recursive-doubling reduction method (RCD) developed by Stone [19], and the odd-even, or cyclic, reduction method (OER) developed by Hockney [10] are able to solve an n-dimensional tridiagonal system in $O(\log(n))$ time using n processors. These methods are designed for fine-grain computing. Substructured methods developed by Lawrie and Sameh [14], Wang [27], and Sun, Zhang, and Ni [26] were designed for median-grain and coarse-grain computing (i.e., the case of $p < n$ or $p << n$, where p is the number of processors available). Lawrie and Sameh's algorithm is designed for shared-memory machines; Wang's algorithm is designed for distributed-memory machines; and Sun et al. proposed three different algorithms, each of which may be a better choice depending on the problem and the machine. For compact schemes, the tridiagonal systems have a special structure that consists of diagonal dominance and are almost symmetric Toeplitz. For this special structure, a parallel tridiagonal solver for fine-grain computing, *the simple parallel*

prefix (SPP) algorithm, is proposed in this paper. It shows that compact schemes can be solved efficiently on parallel computers. Since the same special structure appears in many other scientific applications, such as alternating direction implicit method [21], wavelet collocation method [2], spline curve fitting [4], etc., the importance of the SPP algorithm is beyond compact schemes.

The SPP method is simple to implement. It requires only $2 \cdot \log(n)$ AXPY (vector plus scalar times vector) operations and prefix communication patterns. If the tridiagonal system is diagonally dominant, then the AXPY operations can be truncated after a certain number of steps without degrading the accuracy. A formal accuracy analysis is conducted and simple formulas are provided to compute the number of AXPY operations necessary.

This paper is organized as follows. Section 2 will present the compact scheme and discretized tridiagonal systems. Section 3 will introduce three versions of the simple parallel prefix algorithm: the SPP for tridiagonal systems with the given special structure, the SPP for solving symmetric Toeplitz tridiagonal systems, and the SPP for solving almost symmetric Toeplitz systems. Accuracy analysis will be conducted in Section 4. Section 5 will give experimental results on a 16K processing elements (PEs) MasPar SIMD computer and on a Cray 2 supercomputer. Finally, Section 6 will give the conclusions.

2 Compact Finite-Difference Schemes

With either conventional finite-difference or finite-element discretization methods, as the order of the approximation increases, the required number of boundary and near-boundary relations and the required number of mesh points per derivative stencil increases accordingly. To achieve higher accuracy with less additional mesh points, the compact scheme [12] was introduced. As originally suggested by Kreiss and Oliger [12], and later discussed for fluid dynamics problems by Hirsh [8], the first and second derivatives for compact differences may be approximated by

$$f'_n = \left(\frac{D_0}{1 + \frac{1}{6} h_x^2 D_+ D_-} \right) f_n \quad \text{and} \quad f''_n = \left(\frac{D_+ D_-}{1 + \frac{1}{12} h_x^2 D_+ D_-} \right) f_n, \tag{1}$$

where

$$D_0 f_n = \frac{1}{2h_x}(f_{n+1} - f_{n-1}), \quad D_+ f_n = \frac{1}{h_x}(f_{n+1} - f_n),$$

$$D_- f_n = \frac{1}{h_x}(f_n - f_{n-1}),$$

and h_x is the mesh spacing, which is constant for simplicity. By multiplying (1) by the respective denominators, relations for the derivatives may be found, which yield

$$\frac{1}{6} f'_{n-1} + \frac{2}{3} f'_n + \frac{1}{6} f'_{n+1} = \frac{1}{2h_x}(f_{n+1} - f_{n-1}), \tag{2}$$

and

$$\frac{1}{12}f''_{n-1} + \frac{5}{6}f''_n + \frac{1}{12}f''_{n+1} = \frac{1}{h_x^2}(f_{n+1} - 2f_n + f_{n-1}). \tag{3}$$

These equations yield tridiagonal systems when the appropriate boundary conditions are applied.

To make an accurate comparison between the compact-difference (equations (2) and (3)) and the standard central-difference schemes, Taylor-series expansions are employed. As Hirsh has shown, the truncation error for the compact differences are

$$E(f'_n) = -\frac{1}{180}h_x^4 f^{(v)} \quad \text{and} \quad E(f''_n) = -\frac{1}{240}h_x^4 f^{(vi)}.$$

Similar error analyses for the central differences yield

$$E(f'_n) = -\frac{1}{30}h_x^4 f^{(v)} \quad \text{and} \quad E(f''_n) = -\frac{1}{90}h_x^4 f^{(vi)}.$$

Although both schemes are fourth-order accurate, the compact-difference scheme should lead to more accurate approximations as a result of the smaller coefficients on the truncation error. Similar results hold for other higher order approximations.

As yet, no mention has been made about the boundary treatment for the compact scheme. At the boundaries, Hirsh [8] experimented with a variety of boundary conditions, and Adams [1] suggested a boundary relation that includes near-boundary derivatives in the formulation. The boundary conditions used by both Hirsh and Adams retained the tridiagonal nature of the system. Demonstrated some 30 years ago, the boundary-condition stencil can take the form

$$c_0 f'_0 + f'_1 = \alpha f_0 + \beta f_1 + \gamma f_2 + \varepsilon f_3, \tag{4}$$

where $c_0 = 1$, $\alpha = \frac{-2}{h_x}$, $\beta = -\alpha$, and $\gamma = \varepsilon = 0$ for a second-order boundary condition; $c_0 = \frac{1}{2}$, $\alpha = \frac{-5}{4h_x}$, $\beta = \frac{1}{h_x}$, $\gamma = \frac{1}{4h_x}$, and $\varepsilon = 0$ for a third-order boundary condition; and $c_0 = \frac{1}{3}$, $\alpha = \frac{-17}{18h_x}$, $\beta = \gamma = \frac{1}{2h_x}$, and $\varepsilon = \frac{-1}{18h_x}$ for a fourth-order boundary condition. Additional high order stencils have been described by Carpenter, Gottlieb, and Abarbanel [3]. To demonstrate the SPP algorithm, for simplicity, explicit fourth-order one-sided finite differences will be used for boundary conditions.

Many relevant fluid dynamics applications can make use of high-order compact-difference operators to numerically solve the governing systems of equations. For example, Burger's equation, the boundary-layer equations, and the driven cavity problem were solved by Hirsh [8] with compact-difference operators. Further, Joslin et al. [11] used the compact-difference equations (2) and (3) to numerically solve the fully nonlinear Navier-Stokes equations of fluid dynamics.

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{5}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j}, \tag{6}$$

where $u_i = (u_1, u_2, u_3)$ are the velocity components that correspond to the steamwise, normal, and spanwise directions, $x_i = (x_1, x_2, x_3)$; ν is the kinematic viscosity, ρ is the density, and repeated indices infer a summation over the index. To demonstrate how compact-difference operators could be employed to solve the nonlinear PDE's (5) and (6), consider the model problem of the one-dimensional heat-conduction equation:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{7}$$

To solve this equation computationally, discretizations in time and space must be chosen. From the Taylor-series expansions in time, one derives the discrete equation

$$u^{n+1} = u^n + \kappa \triangle t \frac{\partial^2 u^n}{\partial x^2}, \tag{8}$$

where n corresponds to a time level. If the result at level u^n is known, then the solution u^{n+1} can be obtained if $\partial^2 u^n / \partial x^2$ can be determined. The spatial derivative can be computed with the second-derivative operator (3). Each time-step advancement requires a single compact-difference solve. In the original nonlinear PDE systems (5) and (6), an explicit or semi-implicit time discretization will lead to both first and second order derivative operators evaluated on previous time-step information. On standard cartesian grids, the resulting discrete system requires differential operator solves which are scalar matrices and are similar to the model problem (8). Because time-marching is necessary, compact-difference solves are required at each time level. This necessitates a fast compact-difference solver. By observation, equations (2) and (3) take the matrix form

$$\begin{bmatrix} c_0 & 1 & & & \\ 1 & c & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & c & 1 \\ & & & 1 & c_0' \end{bmatrix} x = d \tag{9}$$

where $x = \{f', f''\}$ and $c = \{4, 10\}$ correspond to the compact-difference parameters. The first and last rows of equation (9) arise from boundary conditions.

With higher orders of approximation, the resulting matrix will differ only in the boundary conditions, however, the resulting tridiagonal systems can be written in the almost symmetric Toeplitz form described in the next section, Eqn. (10), where A is given by (12).

3 The Simple Parallel Prefix Algorithm

We are interested in solving a tridiagonal linear system of equations

$$Ax = d. \tag{10}$$

In this system of equations, A is either a symmetric Toeplitz tridiagonal system of order n

$$A = \begin{bmatrix} c & 1 & & & \\ 1 & c & 1 & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & 1 \\ & & & 1 & c \end{bmatrix} = [1, c, 1], \tag{11}$$

or an almost symmetric Toeplitz tridiagonal system of order n

$$A = \begin{bmatrix} c_0 & 1 & & & \\ 1 & c & 1 & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & 1 \\ & & & 1 & c_0' \end{bmatrix}, \tag{12}$$

where $x = (x_1, \cdots, x_n)^T$ and $d = (d_1 \cdots, d_n)^T$ are n-dimensional vectors. We assume that matrix A is diagonally dominant (i.e., $|c| > 2$). Although we assume that A, x, and d have real coefficients, the extension to the complex case is straightforward.

3.1 The Simple Prefix Method

Before solving symmetric systems and almost symmetric Toeplitz systems with two boundary conditions, the simple prefix method is first introduced to solve a special kind of one-boundary-condition systems typical of hyperbolic systems. The simple prefix method. then, can be modified for more general situations. In this section, we study how to efficiently solve the system

$$\tilde{A}\tilde{x} = d \tag{13}$$

on parallel computers, where

$$\tilde{A} = \begin{pmatrix} a & 1 & & \\ 1 & c & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & c \end{pmatrix} = a \begin{pmatrix} 1 & & & \\ b & 1 & & \\ & \ddots & \ddots & \\ & & b & 1 \end{pmatrix} \begin{pmatrix} 1 & b & & \\ & \ddots & \ddots & \\ & & \ddots & b \\ & & & 1 \end{pmatrix} = a \cdot [b,1,0] \cdot [0,1,b],$$

and a and b are the real solutions of:

$$\begin{cases} a + b = c \\ a \cdot b = 1 \end{cases} \tag{14}$$

For a given value of c, the relations in (14) define the values of a and b. In general, the value obtained for a would lead to an inconsistency with the coefficients of the boundary stencils given in equation (4). However, for the fourth-order compact-difference scheme described here, which will require at least third-order accurate boundary conditions, the third and fourth order boundary conditions shown in equation (4) can be combined to yield a one-parameter family of boundary conditions,

$$af_0' + f_1' = \frac{a}{6h_x}\left(-(15+2\sigma)f_0 + (12-3\sigma)f_1 + (3+6\sigma)f_2 - \sigma f_3\right),$$

where $a = \frac{1}{2(1-\sigma)}$. The value of a is used to determine σ. Note, $\sigma = 0$ and $\sigma = 1$ correspond to the original third and fourth order conditions of equation (4). Because $a \cdot b = 1$ and $|c| > 2$, we may further assume that $|a| > 1$ and $|b| < 1$. By equation (13),

$$\tilde{x} = a^{-1} \cdot [0,1,b]^{-1}[b,1,0]^{-1} d = b \cdot [0,1,b]^{-1}[b,1,0]^{-1} d.$$

Let $L = [-b,0,0]$. Then

$$[b,1,0] = [0,1,0] - [-b,0,0] = I - L$$

and

$$[b,1,0]^{-1} = (I + L + L^2 + \cdots + L^{n-1}) \tag{15}$$

$$= (I + L^{2^{\lceil \lg n \rceil - 1}})(I + L^{2^{\lceil \lg n \rceil - 2}}) \cdots\cdots (I + L^4)(I + L^2)(I + L). \tag{16}$$

Note that n is the dimension of matrix $\tilde{A}$ and that equations (15) and (16) can be verified directly.

The superscript of matrix L represents matrix multiplication

$$L^i = \begin{pmatrix} 0 & & & & \\ 0 & 0 & & & \\ (-b)^i & 0 & 0 & & \\ 0 & \ddots & \ddots & \ddots & \\ 0 & 0 & (-b)^i & 0 & 0 \end{pmatrix},$$

where the first nonzero element $(-b)^i$ is at position $(i+1, 1)$. Similarly, let $U = [0, 0, -b]$. Then,

$$[0, 1, b] = [0, 1, 0] - [0, 0, -b] = I - U,$$

and

$$\begin{aligned} [0, 1, b]^{-1} &= (I + U + U^2 + \cdots + U^{n-1}) \\ &= (I + U^{2^{\lceil \lg n \rceil - 1}}) \cdot \cdots \cdot (I + U^2)(I + U), \end{aligned}$$

where

$$U^i = \begin{pmatrix} 0 & 0 & (-b)^i & 0 & 0 \\ & \ddots & \ddots & \ddots & 0 \\ & & 0 & 0 & (-b)^i \\ & & & 0 & 0 \\ & & & & 0 \end{pmatrix}.$$

The first nonzero element of U^i is at position $(1, i+1)$. Thus, the solution of equation (13) is

$$\tilde{x} = b \cdot (I + U^{2^{\lceil \lg n \rceil - 1}}) \cdots (I + U) \cdot (I + L^{2^{\lceil \lg n \rceil - 1}}) \cdot \cdots \cdot (I + L)d. \tag{17}$$

Let $v = (v_1, v_2, \cdots, v_n)^T$ be an n-dimensional vector. Given the special structure of L^i, we find

$$(I + L^i)v = v + (-b)^i v_{(i)},$$

where

$$v_{(i)} = (0, \cdots 0, v_1, \cdots v_{n-i})^T$$

and v_1 is the $i+1$ element of $v_{(i)}$. Similarly, given the special structure of U^i, we find

$$(I + U^i)v = v + (-b)^i v^{(i)},$$

where

$$v^{(i)} = (v_{i+1}, \cdots, v_n, 0 \cdots 0)^T.$$

Equation (17) shows that equation (13) can be solved with $2 \cdot \lceil \lg n \rceil$ AXPY operations. Because $|b| < 1$, $||L^i|| \to 0$ and $||U^i|| \to 0$ when $n \to \infty$, the AXPY operation may be truncated without influencing the accuracy. Formulas will be given in Section 4 to compute the smallest truncation integer $\bar{k}$. A sequential code for solving equation (17) within truncation error is given in figure 1.

```
for  i ← 0  to  k̄ − 1  do
  for  j ← 2^i  to  n  do
    d_j = d_j + (−b)^{2^i} d_{j−2^i}
    j ← j + 1
  i ← i + 1

for  i ← 0  to  k̄ + 1  do
  for  j ← 1  to  n − 2^i  do
    d_j = d_j + (−b)^{2^i} d_{j+2^i}
    j ← j + 1
  i ← i + 1

for  j ← 1  to  n  do
  x̃_j = b · d_j
  j ← j + 1
```

Figure 1. The simple prefix method.

If n processing elements are available and d_j is stored in processor j, then the two `for loop`s of i in figure 1 will lead to prefix computations. Figure 2 shows the prefix computation pattern that correspond to the second `for loop` of i when n is equal to 8. The first `for loop` of i in figure 1 leads to a similar prefix computation pattern, except that the communication is from right to left. Prefix (or recursive doubling) computation is a widely used computation model in scientific computing. Any linear recursive relation can be computed by recursive doubling [23]. A recursive-doubling algorithm exists for solving tridiagonal systems and involves matrix-matrix multiplications [6, 26]. Compared with the existing recursive-doubling algorithm, our method achieves a smaller operation count by adopting the "vertical prefix" computation, in which the matrix-vector multiplication in equation (17) is conducted in a parallel prefix fasion. Compared with the cyclic reduction method [10], the proposed prefix method has a simpler communication pattern. We call the prefix method given in figure 1 the simple prefix method. Figure 3 shows the communication pattern of the widely used cyclic reduction method [10]. In a comparison of figures 2 and 3, we can see that the prefix

method has a relatively simple communication pattern. Both of the SPP algorithm and the cyclic reduction method need $2 \times log(n)$ communication and computation steps, and both can truncate these steps if the system is diagonally dominant. The SPP algorithm has a slightly lower bound than that of the existing algorithm on the solution accuracy for the truncated algorithm, in the case of diagonally dominant systems. On the other hand, the proposed prefix method also has its limitation. It is only feasible for solving almost Toeplitz tridiagonal systems.

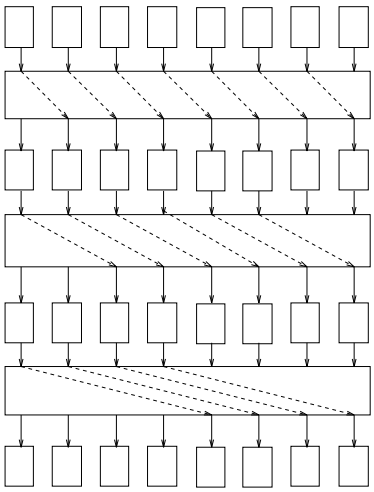

Figure 2. Communication of prefix computation.

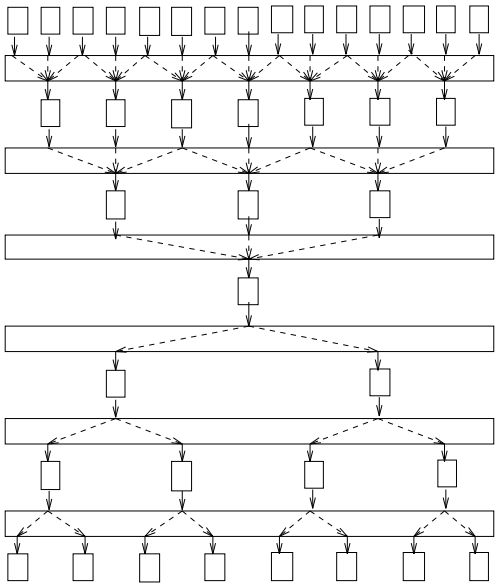

Figure 3. Communication of cyclic reduction method.

Fast sine transform (FST), which can be implemented efficiently on certain parallel machines, can also be used to solve symmetric Toeplitz tridiagonal systems. The FST method has a similar

communication pattern (see Fig. 4) and similar computation count[1] as the SPP algorithm on fine-grain parallel computers. However, the communication and computation of the FST method cannot be truncated even the linear system is diagonal dominant.

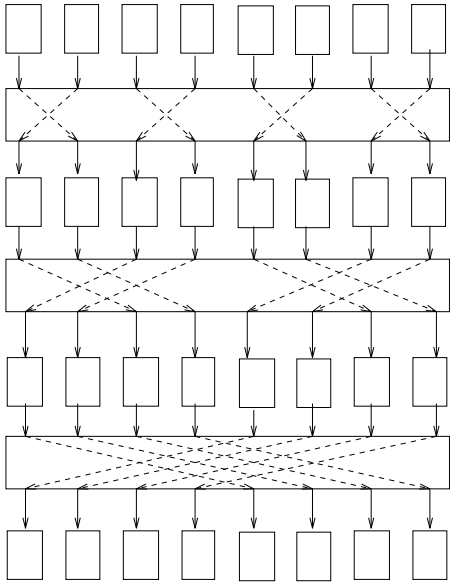

Figure 4. Communication of Fast sine transform method.

3.2 Modification of Symmetric Toeplitz System

Our goal is to find the solution of equation (10). Modification is needed to convert the solution of equation (13) to the desired solution. The modification will be different for symmetric Toeplitz systems and for almost symmetric Toeplitz systems. For a given symmetric Toeplitz system, equation (13) is modified as

$$A = \tilde{A} + \triangle A = \tilde{A} + VE^T,$$

where

$$\triangle A = \begin{pmatrix} b & 0 & & & & \\ 0 & 0 & 0 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 0 & 0 & 0 & \\ & & & 0 & 0 & \end{pmatrix}, \tag{18}$$

[1]Here we assume that the n unit roots are readily available. Otherwise, the FST method will have a high initialization time.

and $V = (b, 0, \cdots, 0)^T$ and $E = (1, 0, \cdots, 0)^T$ are n-dimensional vectors. By the matrix-modification formula [18, 5, 26], equation (10) can be solved by

$$\begin{aligned} x &= A^{-1}d = (\tilde{A} + VE^T)^{-1}d \\ x &= \tilde{A}^{-1}d - \tilde{A}^{-1}V(I + E^T\tilde{A}^{-1}V)^{-1}E^T\tilde{A}^{-1}d \\ &= \tilde{x} - \tilde{A}^{-1}V(I + E^T\tilde{A}^{-1}V)^{-1}\tilde{x}_1, \end{aligned}$$

where $\tilde{x}_1$ is the first element of vector $\tilde{x}$. If the calculation approach given in [21] is followed, we have

$$(I + E^T\tilde{A}^{-1}V)^{-1} = \frac{1}{1 + \sum_{i=1}^{n} b^{2i}}$$

and

$$\tilde{A}^{-1}V = (\sum_{i=1}^{n} b^{2i}, \sum_{i=1}^{n-1}(-b)b^{2i}, \sum_{i=1}^{n-2}(-b)^2 b^{2i}, \cdots, (-b)^{n+1})^T.$$

Thus

$$\tilde{A}^{-1}V(I + E^T\tilde{A}^{-1}V)^{-1}$$

$$= b^2 \left(\frac{1 - b^{2n}}{1 - b^{2(n+1)}}, (-b)\frac{1 - b^{2(n-1)}}{1 - b^{2(n+1)}}, \cdots, (-b)^i \frac{1 - b^{2(n-i)}}{1 - b^{2(n+1)}}, \cdots, (-b)^{n-1}\frac{(1 - b^2)}{1 - b^{2(n+1)}} \right)^T. \qquad (19)$$

The final solution is

$$x = \tilde{x} - \tilde{x}_1 z, \qquad (20)$$

where vector z is the right side of equation (19). Because $|b| < 1$, z can be truncated at some integer k_1 without affecting the accuracy (see section 4). Furthermore, when n is large, $b^{2(n-i+1)}$, $i = 0, 1, \cdots, k_1$, will be less than machine accuracy, and z reduces to

$$\tilde{z} = ((-b)^2, (-b)^3, \cdots, (-b)^{k_1+2}, 0, \cdots, 0)^T.$$

The program of modification is given in Figure 5, and the algorithm for solving the symmetric Toeplitz tridiagonal system is given in Figure 6.

for $i \leftarrow 1$ **to** k_1 **do**
$x = \tilde{x}_i - \tilde{x}_1\tilde{z}_i$
$i \leftarrow i + 1$

Figure 5. Modification for symmetric Toeplitz systems.

Step 1:

Use the simple prefix method to find the solution to equation (13).

Step 2:

Use the modification equation (20) to obtain the final solution.

Figure 6. Algorithm for solving symmetric Toeplitz system.

3.3 Modification of Almost Symmetric Toeplitz System

A similar modification can be given to solve almost symmetric Toeplitz systems. For almost symmetric Toeplitz systems, equation (13) is modified such that

$$A = \tilde{A} + \triangle\tilde{A} = \tilde{A} + \tilde{V}\tilde{E}^T,$$

where

$$\triangle\tilde{A} = \begin{pmatrix} c_1 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & c_2 \end{pmatrix}, \tag{21}$$

$c_1 = c_0 - a$, $c_2 = c_0' - c$, and

$$\tilde{V} = \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & ... & 0 & 1 \end{pmatrix}^T, \qquad \tilde{E} = \begin{pmatrix} c_1 & 0 & ... & 0 \\ 0 & ... & 0 & c_2 \end{pmatrix}^T,$$

Following the matrix modification formula [26, 21], the solution to equation (10) becomes

$$\begin{aligned} x &= A^{-1}d = \tilde{A}^{-1}d - \tilde{A}^{-1}\tilde{V}(I + \tilde{E}^T\tilde{A}^{-1}\tilde{V})^{-1}\tilde{E}^T\tilde{A}^{-1}d \\ &= \tilde{x} - \tilde{A}^{-1}\tilde{V}(I + \tilde{E}^T\tilde{A}^{-1}\tilde{V})^{-1}\begin{pmatrix} c_1\tilde{x}_1 \\ c_2\tilde{x}_n \end{pmatrix}, \end{aligned}$$

where $\tilde{x}_1$ and $\tilde{x}_n$ are the first and last element of $\tilde{x}$, respectively. With this new modification,

$$\begin{aligned}\tilde{A}^{-1}\tilde{V} &= \begin{pmatrix} \sum_{i=0}^{n-1} b^{2i+1} & \sum_{i=0}^{n-2} -b^{2i+2} & \cdots & \sum_{i=0}^{n-j}(-1)^{j-1}b^{2i+j} & \cdots & (-1)^{n-1}b^n \\ (-1)^{n-1}b^n & (-1)^{n-2}b^{n-1} & \cdots & (-1)^{n-j}b^{n-j+1} & \cdots & b \end{pmatrix}^T \\ &= \begin{pmatrix} y^1 \\ y^2 \end{pmatrix}^T = y^T.\end{aligned}$$

$$(I + \tilde{E}^T\tilde{A}^{-1}\tilde{V})^{-1} = \begin{pmatrix} 1 + c_1 b\frac{1-b^{2n}}{1-b^2} & (-1)^n c_1 b^n \\ (-1)^n c_2 b^n & 1 + c_2 b \end{pmatrix}^{-1}.$$

The final solution is

$$x = \tilde{x} - b_1 y^1 - b_2 y^2, \tag{22}$$

where $b_1,\ b_2$ are the solutions of the reduced 2×2 system.

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = (I + \tilde{E}^T\tilde{A}^{-1}\tilde{V})^{-1} \begin{pmatrix} c_1\tilde{x}_1 \\ c_2\tilde{x}_n \end{pmatrix}. \tag{23}$$

Similar to the symmetric case, when n is large, $|b|^n$ may be less than machine accuracy. In these cases,

$$(I + \tilde{E}^T\tilde{A}^{-1}\tilde{V})^{-1} = \begin{pmatrix} 1 + \frac{c_1}{a-b} & \\ & 1 + c_2 b \end{pmatrix}^{-1},$$

and

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} \frac{(a-b)c_1}{a-b+c_1}\tilde{x}_1 \\ \frac{c_2}{1+c_2 b}\tilde{x}_n \end{pmatrix}.$$

Thus, when n is large, the final solution can be computed through a simplified form,

$$x = \tilde{x} - \frac{c_1\tilde{x}_1}{a - b + c_1}\tilde{y}^1 - \frac{c_2\tilde{x}_n}{1 + c_2 b}\tilde{y}^2 \tag{24}$$

where

$$\begin{pmatrix} \tilde{y}^1 \\ \tilde{y}^2 \end{pmatrix}^T = \begin{pmatrix} 1 & -b & \cdots & (-b)^{j-1} & \cdots & (-b)^{n-1} \\ (-1)^{n-1}b^n & (-1)^{n-2}b^{n-1} & \cdots & (-1)^{n-j}b^{n-j+1} & \cdots & b \end{pmatrix}^T.$$

The modification computation for the almost symmetric Toeplitz system is given in figure 7. The algorithm for solving the almost symmetric Toeplitz systems is similar to the algorithm for solving the symmetric Toeplitz system (see figure 6), except in step 2 the new modification (figure 7) is used to replace the symmetric Toeplitz system modification. Notice that when $c_0 = a$ and $c_0' = c$, we have $c_1 = c_2 = 0$ and there is no modification necessary. When $c_0 = c$ and $c_0' = c$, we

have $c_1 = b$, $c_2 = 0$, and figure 7 is equivalent to figure 5.

for $i \leftarrow 1$ **to** n **do**

$x = \tilde{x}_i - \frac{c_1 \tilde{x}_1}{a-b+c_1}\tilde{y}_i^1 - \frac{c_2 \tilde{x}_n}{1+c_2 b}\tilde{y}_i^2$

$i \leftarrow i + 1$

Figure 7. Modification for almost symmetric Toeplitz system.

4 Accuracy Analysis

In a previous section, the claim was made that the AXPY operations and modifications can be truncated without influencing the accuracy. In this section, we will study the truncation accuracy of symmetric Toeplitz systems. For simplicity, we truncate the first `for loop` and the second `for loop` at the same step $\bar{k}$. The norm used in this section is the 1-norm.

4.1 Accuracy Study of the Simple Prefix Method

Let $\tilde{x}, y$ be exact solutions as

$$\begin{aligned} y &= b \cdot [b, 1, 0]^{-1} d, \\ &= (I + L + L^2 + \cdots + L^{n-1}) b \cdot d, \\ \tilde{x} &= b \cdot [0, 1, b]^{-1} [b, 1, 0]^{-1} \cdot d \\ &= (I + U + U^2 + \cdot + U^{n-1}) \cdot y. \end{aligned}$$

Let y^* and $\tilde{x}'$ be the solutions given by the first `for loop` and the second `for loop`, respectively, as

$$y^* = (I + L + \cdots + L^{k-1}) \cdot b \cdot d,$$

where k is a power of 2 ($k = 2^{\bar{k}}$), and

$$\tilde{x}' = (I + U + \cdots + U^{k-1}) y^*.$$

Also, we define $\tilde{x}^*$ as a hypothetical solution by

$$\tilde{x}^* = (I + U + \cdots + U^{k-1}) y.$$

The difference between $\tilde{x}$ and $\tilde{x}^*$ is

$$\begin{aligned}\tilde{x} - \tilde{x}^* &= (I + U + \cdots + U^{n-1})y - (I + U + \cdots + U^{k-1})y \\ &= (U^k + U^{k+1} + \cdots + U^{n-1})y \\ &= (U^k + \cdots + U^{n-1})(I - U)\tilde{x} \\ &= (U^k - U^n)\tilde{x}.\end{aligned}$$

Thus, we find

$$\frac{||\tilde{x} - \tilde{x}^*||}{||\tilde{x}||} \leq ||U^k|| \cdot ||I - U^{n-k}|| \leq |b|^k(1 + |b|^{n-k}). \tag{25}$$

Equation (25) gives the relative error between $\tilde{x}$ and $\tilde{x}^*$. Because the difference between $\tilde{x}^*$ and $\tilde{x}'$ is

$$\begin{aligned}\tilde{x}^* - \tilde{x}' &= (I + U + \cdots + U^{k-1})y - (I + U + \cdots + U^{k-1})y^* \\ &= (I + U + \cdots + U^{k-1})(y - y^*) \\ &= (I + U + \cdots + U^{k-1})(L^k + \cdots + L^{n-1}) * b \cdot d \\ &= (I + U + \cdots + U^{k-1})(L^k + \cdots + L^{n-1})(I - L)(I - U)\tilde{x} \\ &= (I + \cdots + U^{k-1})(L^k - L^n)(I - U)\tilde{x},\end{aligned}$$

the norm becomes

$$\frac{||\tilde{x}^* - \tilde{x}'||}{||\tilde{x}||} \leq \frac{1 - |b|^k}{1 - |b|} \cdot |b|^k \cdot (1 + |b|^{n-k})(1 + |b|). \tag{26}$$

If equations (25) and (26) are combined, then the relative error between the exact solution $\tilde{x}$ and the truncated solution $\tilde{x}'$ is

$$\begin{aligned}\frac{||\tilde{x} - \tilde{x}'||}{||\tilde{x}||} &\leq \frac{||\tilde{x} - \tilde{x}^*||}{||\tilde{x}||} + \frac{||\tilde{x}^* - \tilde{x}'||}{||\tilde{x}||} \\ &\leq |b|^k(1 + |b|^{n-k})(1 + \frac{1 - |b|^k}{1 - |b|}(1 + |b|)) \\ &\leq 2 \cdot |b|^k \cdot (1 + |b|^{n-k}).\end{aligned}$$

4.2 Final Accuracy of Symmetric Toeplitz Systems

The truncation error of the simple prefix method (Figure 1) will be carried into the modification step and will influence the accuracy of the final solution. Let x be the solution of a symmetric Toeplitz tridiagonal system (10) and x^* be the corresponding solution that has been modified with the truncated solution,

$$\begin{aligned}x &= \tilde{x} - \tilde{x}_1 z, \\ x^* &= \tilde{x}' - \tilde{x}'_1 z,\end{aligned}$$

where $z = b^2 \left(\frac{1-b^{2n}}{1-b^{2(n+1)}}, (-b)\frac{1-b^{2(n-1)}}{1-b^{2(n+1)}}, \cdots (-b)^{n-1}\frac{(1-b^2)}{1-b^{2(n+1)}} \right)^T$; see equation (19). The error generated by this truncation is

$$\begin{aligned} ||x - x^*|| &= ||(\tilde{x} - \tilde{x}') + (\tilde{x}_1 - \tilde{x}_1')z|| \\ &\leq ||\tilde{x} - \tilde{x}'|| + ||\tilde{x}_1 - \tilde{x}_1'|| \cdot ||z|| \\ &\leq ||\tilde{x} - \tilde{x}'||(1 + ||z||). \end{aligned}$$

The norm of vector z can be computed directly as

$$\begin{aligned} ||z|| &= \frac{b^2}{1-b^{2(n+1)}} \sum_{i=0}^{n-1} |(b)^i(1-b^{2(n-i)})| \\ &\leq \frac{b^2}{1-b^{2(n+1)}} (\sum_{i=0}^{n-1} |b|^i + \sum_{i=0}^{n-1} |b|^{2n-i}) \\ &\leq \frac{b^2}{1-b^{2(n+1)}} \cdot \frac{(1+|b|^{n+1})(1-|b|^n)}{(1-|b|)} \\ &\leq b^2 \cdot \frac{(1-|b|^n)}{(1-|b|^{n+1})(1-|b|)} \leq \frac{b^2}{1-|b|} = \frac{|b|}{|a|-1}. \end{aligned}$$

Therefore,

$$\begin{aligned} ||x - x^*|| &\leq ||\tilde{x} - \tilde{x}'||(1 + \frac{|b|}{|a|-1}) && (27) \\ &= ||\tilde{x} - \tilde{x}'||(\frac{|c|-1}{|a|-1}) && (28) \\ &= ||\tilde{x} - \tilde{x}'||(\frac{1-|b|+|b|^2}{1-|b|}) \leq ||\tilde{x} - \tilde{x}'||(\frac{1}{1-|b|}), && (29) \end{aligned}$$

and

$$\begin{aligned} \frac{||\tilde{x} - \tilde{x}^*||}{||x||} &\leq \frac{||\tilde{x} - \tilde{x}'||}{||x||}(\frac{1-|b|+|b|^2}{1-|b|}) && (30) \\ &= \frac{||\tilde{x} - \tilde{x}'||}{||\tilde{x}||} \cdot \frac{||\tilde{x}||}{||x||}(\frac{1-|b|+|b|^2}{1-|b|}). && (31) \end{aligned}$$

The only unknown in the right side of equation (31) is $\frac{||\tilde{x}||}{||x||}$, where $\tilde{x}$ is the solution of equation (13) and x is the solution of equation (10). For a symmetric Toeplitz system,

$$A = \tilde{A} + \triangle A,$$

where $\triangle A$ is given by equation (18). If equations (10) and (13) are combined, we have

$$(\tilde{A} + \triangle A)x = \tilde{A}\tilde{x},$$

$$(I + \tilde{A}^{-1}\triangle A)x = \tilde{x},$$

and

$$\frac{||\tilde{x}||}{||x||} \leq ||I + \tilde{A}^{-1}\triangle A||. \tag{32}$$

After several calculations, we find

$$\tilde{A}^{-1} = b \begin{pmatrix} 1 & -b & b^2 & \cdot & (-b)^{n-1} \\ & 1 & -b & \cdot & (-b)^{n-2} \\ & & \cdot & \cdot & \cdot \\ & & & 1 & -b \\ & & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & & \\ -b & 1 & & & \\ b^2 & -b & 1 & & \\ \cdot & \cdot & \cdot & \cdot & \\ (-b)^{n-1} & \cdot & \cdot & -b & 1 \end{pmatrix}$$

and

$$\tilde{A}^{-1}\triangle A = \begin{pmatrix} \sum_{i=1}^{n} b^{2i} & 0 & \cdot & \cdot & 0 \\ \sum_{i=1}^{n-1}(-b)b^{2i} & 0 & \cdot & \cdot & 0 \\ \sum_{i=1}^{n-2}(-b)^2 b^{2i} & 0 & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ (-b)^{n+1} & 0 & \cdot & \cdot & 0 \end{pmatrix}.$$

Therefore,

$$\begin{aligned} ||\tilde{A}^{-1}\triangle A|| &\leq \sum_{i=0}^{n-1} \left| (-b)^i \frac{b^2(1-b^{2(n-i)})}{1-b^2} \right| \\ &\leq \frac{b^2}{1-b^2} \sum_{i=0}^{n-1} |b^i(1-b^{2(n-i)})| \\ &\leq \frac{b^2}{1-b^2} \cdot \frac{(1+|b|^{n+1})(1-|b|^n)}{(1-|b|)} \end{aligned}$$

and

$$||I + \tilde{A}^{-1}\triangle A|| \leq 1 + \frac{b^2}{(1-b^2)} \cdot \frac{(1+|b|^{n+1})(1-|b|^n)}{(1-|b|)}.$$

The relative error of the solution to equation (10) is

$$\frac{||x - x^*||}{||x||} \leq \frac{||\tilde{x} - \tilde{x}'||}{||\tilde{x}||} \cdot \frac{||\tilde{x}||}{||x||} \cdot \frac{1}{1-|b|} \tag{33}$$

$$\leq \frac{|b|^k(1+|b|^{n-k})}{1-|b|} \left(1 + \frac{(1-|b|^k)(1+|b|)}{1-|b|}\right) \left(1 + \frac{b^2(1+|b|^{n+1})(1-|b|^n)}{(1-b^2)(1-|b|)}\right) \tag{34}$$

$$\leq \frac{|b|^k(1+|b|^{n-k})}{1-|b|} \left(1 + \frac{(1-|b|^k)(1+|b|)}{1-|b|}\right) \left(1 + \frac{b^2}{(1-b^2)(1-|b|)}\right). \tag{35}$$

In addition to the truncation error carried from step 1, the truncated modification will also generate truncation error. If

$$x' = \tilde{x}' - \tilde{x}'_1 \tilde{z},$$

(Figures 5 and (20)), then

$$x - x' = \tilde{x} - \tilde{x}' - (\tilde{x}_1 - \tilde{x}'_1)\tilde{z} + \tilde{x}_1(z - \tilde{z}).$$

The 1-norm is given by

$$\begin{aligned} ||x - x'|| &\leq ||\tilde{x} - \tilde{x}'|| + ||\tilde{x} - \tilde{x}'|| \cdot ||\tilde{z}|| + ||x|| \cdot ||z - \tilde{z}|| \\ &\leq ||\tilde{x} - \tilde{x}'||(1 + ||\tilde{z}||) + ||x|| \cdot ||z - \tilde{z}||, \end{aligned}$$

which leads to

$$\frac{||x - x'||}{||x||} \leq \frac{||\tilde{x} - \tilde{x}'||}{||\tilde{x}||} \cdot \frac{||\tilde{x}||}{||x||}(1 + ||\tilde{z}||) + ||z - \tilde{z}||. \tag{36}$$

In equation (36),

$$\begin{aligned} ||z - \tilde{z}|| &= \frac{b^2}{1 - b^{2(n+1)}} \sum_{i=k_1}^{n-1} |b^i(1 - b^{2(n-i)})| \\ &= \frac{b^2}{1 - b^{2(n+1)}} \sum_{j=0}^{n1-1} |b^{i+k_1}(1 - b)^{2(n1-j)})|, \end{aligned}$$

where $j = i - k_1$, $n1 = n - k_1$. Then

$$\begin{aligned} ||z - \tilde{z}|| &\leq \frac{|b|^{k_1+2}}{1 - b^{2(n+1)}} \cdot \frac{(1 + |b|^{n1+1})(1 - |b|^{n1})}{(1 - |b|)}. \\ &\leq \frac{|b|^{k_1+2}}{1 - b^{2(n1+1)}} \frac{(1 + |b|^{n1+1})(1 - |b|^{n1})}{(1 - |b|)} \\ &\leq \frac{|b|^{k_1+2}}{1 - |b|}. \end{aligned}$$

Note that $||\tilde{z}|| \leq ||z||$, so that we have the inequality

$$\frac{||x - x'||}{||x||} \leq \frac{||\tilde{x} - \tilde{x}'||}{||\tilde{x}||} \frac{||\tilde{x}||}{||x||}(1 + ||z||) + \frac{|b|^{k_1+2}}{1 - |b|} \tag{37}$$

$$\leq \frac{|b|^k}{1 - |b|} \left(1 + \frac{(1 - |b|^k)(1 + |b|)}{1 - |b|}\right) \left(1 + \frac{b^2}{(1 - b^2)(1 - |b|)}\right) + \frac{|b|^{k_1+2}}{1 - b}. \tag{38}$$

The error introduced by the truncated modification is insignificant if $k_1 > k - 2$. In practice, we can choose $k_1 = k$ and use the inequality (35) to compute the truncation number k.

5 Experimental Results

In this section, the performance of the SPP algorithm on the MasPar parallel computer and on the Cray vector computer will be presented.

5.1 Parallel Computing

The MasPar MP-1 is a distributed-memory massively parallel SIMD computer with a high-speed two-dimensional toroidal mesh topology. A control unit (ACU) has a direct connection to all the processing elements (PEs) and issues instructions at a 12.5 MHz clock rate. Each processing element in the array is a 4-bit custom load and storage processor with a minimum of 16 kilobytes of memory. Communication is relatively cheap on the MasPar. For example, on the MasPar M-1, a double-precision multiplication function is ten times more expensive than sending the product to an adjacent PE.

Table 5.1 gives the computation and communication count of the simple parallel prefix (SPP) algorithm based on the algorithm (see figure 6) and the communication pattern (see figure 2). Since the SPP algorithm can be used for almost symmetric Toeplize systems, the best sequential algorithm used is the conventional algorithm, Thomas algorithm [20], the LU decomposition method for tridiagonal systems. For symmetric Toeplize systems, a fast method proposed by Malcolm and Palmer [16] only requires $5n + 2k - 3$ arithmetic operations for solving a single system, where k is a decay parameter depending on the diagonal dominancy of the system. The computation savings of Malcolm and Palmer's method is in the LU decomposition. For systems with multiple right hand sides, in which the factorization cost is not considered, the Malcolm and Palmer's method and Thomas method have the same computation count. We assume that the AXPY operations are truncated after $\bar{k}$ operations; the modification vector used is either equation (20) or (38), for symmetric and almost symmetric system, respectively. The $b^{2^i}, i = 1, \cdots, \bar{k}$, are computed sequentially, or redundantly, on each PE. The modification vector $\tilde{z}$ or $\tilde{y}_i$, $i = 1, 2$, will be computed concurrently on different PEs. We count a power-function computation as 8 floating-point operations. So, the modification phase costs 10 parallel operations (20 for almost symmetric systems) in total. The calculation of b (equation (14)) is not considered. In the computing phase, $2 \cdot \log(k) = 2 \cdot \bar{k}$ parallel, one-to-one communications are required. We use α to represent the one-to-one communication. On the MasPar, the one-to-one communication is achieved by using the *router* command. We use β to represent the broadcast. On the MasPar, the broadcast is achieved by transferring the local data to ACU and then distributing it to all PEs. One broadcast communication is needed in the modification phase. Because the tridiagonal systems that arise in the compact scheme have multiple right sides, the computation and communication count for solving multiple right-side systems is also listed in Table 1, where the computation of b^{2^i} and the modification vector are not considered. Note that n_1 is the number of right sides.

Table 1. Computation and communication count of the Simple Prefix Algorithm

System	Best sequential	SPP	
		Computation	Communication
Single system	$8n$-7	$5 \cdot \bar{k} + 10$	$2 \cdot \bar{k} \cdot \alpha + \beta$
Multiple right sides	$(5n-3) * n1$	$(4 \cdot \bar{k} + 2) * n1$	$(2 \cdot \bar{k} \cdot \alpha + n1 * \beta)$

Two sample matrices are chosen to illustrate the performance of the SPP algorithm and to verify the theoretical error bounds given in the previous section. Both of the matrices are symmetric Toeplitz matrices that arise in the compact schemes. One matrix is

$$A_1 = [1, 3, 1] - \triangle\tilde{A}.$$

The other matrix is

$$A_2 = [1, 10, 1] - \triangle\tilde{A}.$$

Equation (18) defines $\triangle\tilde{A}$. The corresponding solution of equation (14) is $b = \frac{3-\sqrt{5}}{2}$ and $b = \frac{7-\sqrt{45}}{2}$ for A_1 and A_2, respectively. The error is measured relative to the LU solution. The accuracy comparison for solving system A_1 is given in figure 8. In this implementation, no truncation is implemented in the modification phase, and the prediction formula used is equation (35). In solving A_2, the modification is applied at the modification stage with $k_2 = k$, and the prediction formula used is equation (38). The accuracy comparison for solving system A_2 is given in figure 9. From figures 8 and 9, we can see that the theoretical bound matches the measured results well.

Speedup is defined as sequential execution time over parallel execution time. Figure 10 shows the speedup of the SPP algorithm over the conventional sequential algorithm for solving a single system. The sequential algorithm, Thomas algorithm, is based on the LU decomposition, and is fine-tuned to take advantage of the almost symmetric Toeplitz structure. All computations are double precision. Truncation numbers $\bar{k} = 6$ and $\bar{k} = 4$ are chosen to achieve double-precision (10^{-16}) and single-precision (10^{-7}) accuracy, respectively. The order of matrix is the same as the number of PEs available. Because of the high-speed communication, with n equal to $1K$, $2K$, $4K$, $8K$, and $16K$ (where $K = 2^{10}$), the execution time is not noticeably changed in parallel processing. The sequential algorithm is implemented on a single PE. Because of memory limitations, only small systems are solved by the sequential algorithm. The data used in figure 10 is predicted based on the small-size timing. Figure 11 shows the corresponding speedup of solving a system with 1024 right sides. The factorization of the matrix is not included in timing for solving the system with multiple right sides. The speedup is slightly higher for solving multiple right-side systems.

Because the order of the matrix increases linearly with the number of PEs available, the speedups given by figures 10 and 11 are memory-bounded speedup [24]. From table 5.1, the problem size,

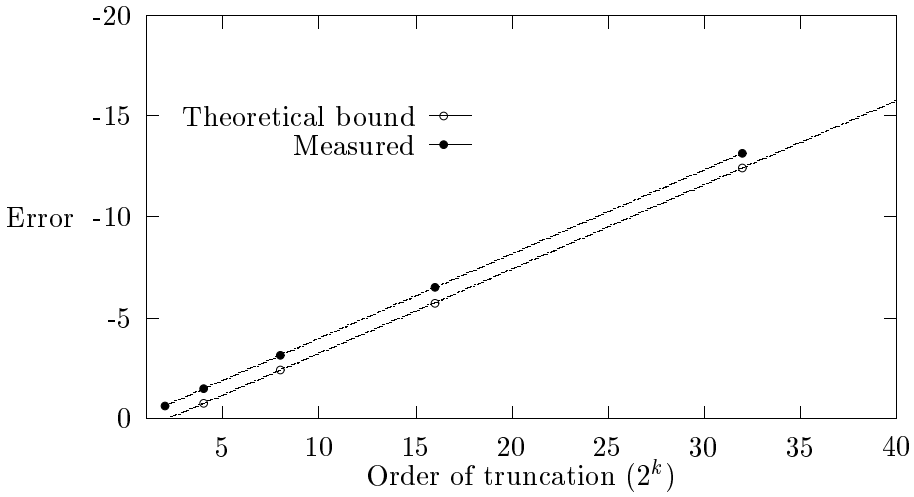

Figure 8. Measured and predicted accuracy for solving matrix A_1.

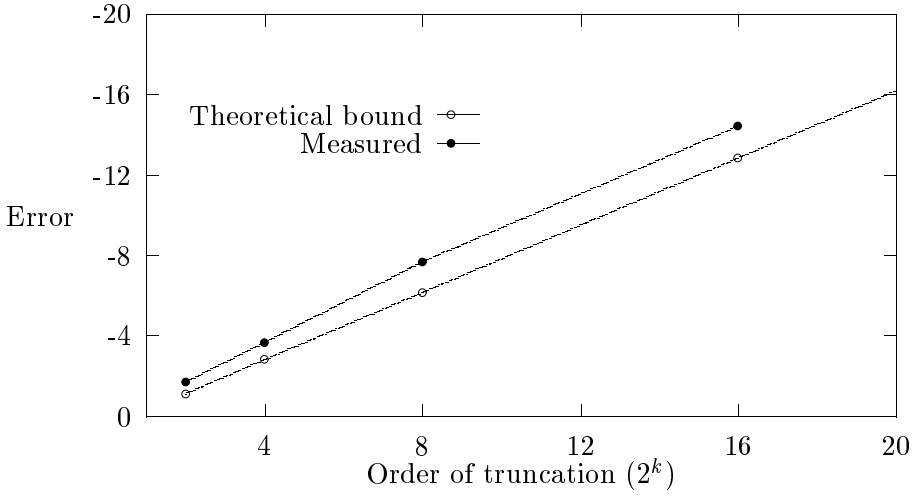

Figure 9. Measured and predicted accuracy for solving matrix A_2.

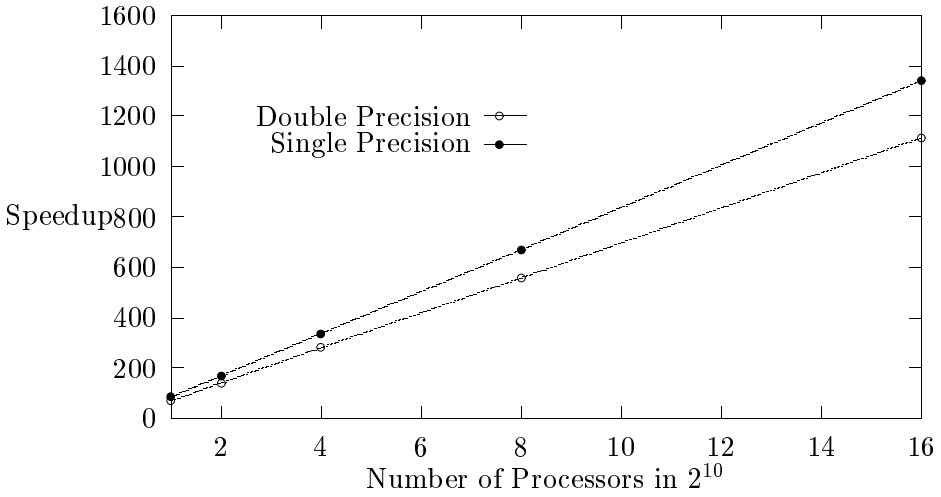

Figure 10. Speedup over the best sequential algorithm on single system.

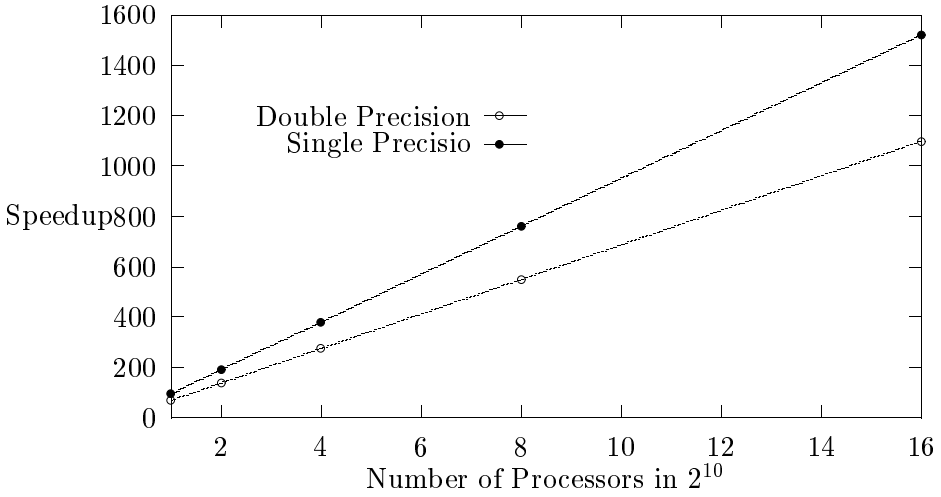

Figure 11. Speedup over the best sequential algorithm on system with 1024 right sides.

in terms of floating-point operations, is a linear function of the order of the matrix. The linear memory-bounded speedups given by figures 10 and 11 indicate a linear speed increase. In accordance with the isospeed metric of scalability [25], the SPP algorithm is perfectly scalable on the SIMD MasPar machine. The reader may refer to [25] for more information regarding scalability of parallel algorithm-machine combinations.

5.2 Vector Computing

Vector computing is widely used at national laboratories, universities, and supercomputing centers for large-scale computing applications. For this reason, a CRAY-2S/4-128 at NASA Langley Research Center was also used to test the SPP algorithm on a vector machine. The Cray-2 notation "S" indicates that the memory is static rather than dynamic, and "4-128" indicates that the machine has 4 processors and 128 million 64-bit words of central memory. Each CPU is a register-to-register vector processor with a 4.1 nsec minor cycle clock that can generate 100-300 megaflops. The four processors can be used for a single problem (multi-tasking) to achieve over 1 gigaflop of performance.

The speed of a vector machine depends on the vector length, vector stride, and the computational richness of the loops. Because the vector register length is 64 and the CPU is extremely fast in carrying out floating-point operations, once operands are in the registers, best performance can be obtained with loop which have lengths that are multiples of 64, which are computationally intensive, and which use unit stride (separation of memory between elements).

The chosen sample tridiagonal matrices are symmetric Toeplitz and correspond to the first and second derivative compact-difference operators (2) and (3). The diagonals and necessary coefficients are:

$$A_3 = [1, 4, 1] \qquad \text{with} \quad a,\ b = 2 \pm \sqrt{3}$$

$$A_2 = [1, 10, 1] \qquad \text{with} \quad a,\ b = 5 \pm 2\sqrt{6}$$

For the first experiment on the Cray, single tridiagonal solves were made. The test problem used here and in the rest of this section corresponds to $f(x) = 3x^3 - 2x + 1$, which has smooth exact derivatives $f'(x) = 9x^2 - 2$ and $f''(x) = 18x$. Figure 12 shows the performance of the SPP in terms of CPU seconds and the matrix order compared with the LU decomposition for computing f' and f''. In addition to the reduced memory requirements of SPP compared to LU, the performance shown in figure 12 clearly indicated that the SPP is faster on the vector machine than the conventional LU solver; the benefits increase with the operator size. The significant difference between the SPP and LU timing can be explained in light of vector operations versus scalar operations. The SPP approach can be vectorized over the direction of the solve; the LU approach must use scalar operations. For the SPP approach, note that the diagonal dominance of

the second-derivative operator f'' leads to faster computations compared with the first-derivative operator f'. This time reduction results from the truncation of the SPP approach to obtain a predetermined level of error (10^{-14}), which is essentially machine precision. For the first-derivative operator (A_3), $k = 32 = 2^6$ and $k_1 = 24$; for the second-derivative operator (A_2), $k = 16 = 2^4$ and $k_1 = 16$, where $k = 2^{\bar{k}}$ and k_1 are the truncation numbers on the solving and modification phases, respectively.

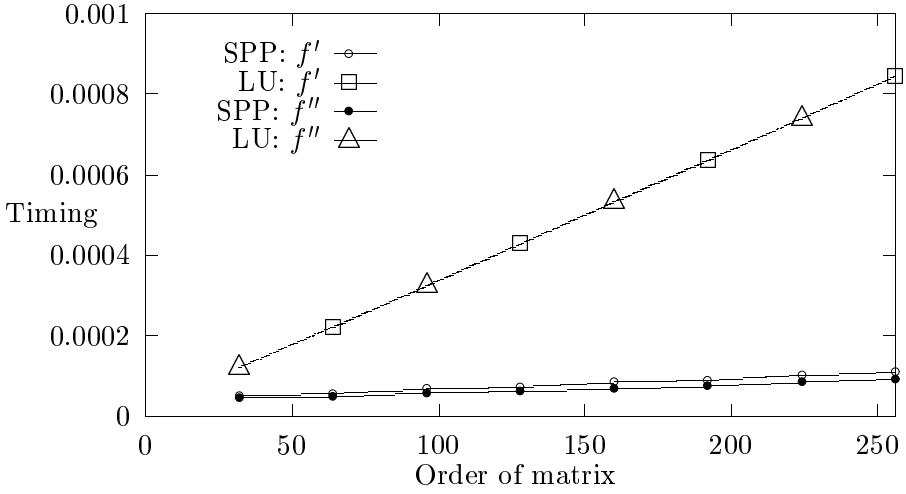

Figure 12. Timing of SPP and LU algorithms: single system.

Real applications which use compact-difference operators require many tridiagonal solves that correspond to time-marching algorithms and involve many right sides corresponding to the multidimensionality of the application. In this second evaluation, with the same accuracy 10^{-14}, the performance of the SPP is compared with LU for multiple right sides. Shown in figure 13 are CPU times for the SPP and LU for various orders of the second-derivative compact operator A_2. (Similar results were obtained with the operator A_3 but are not shown.) For applications that use small operators ($N < 96$), the LU solver is more efficient than SPP; for applications that use large operators ($N > 96$), the SPP is much cheaper than the LU approach. This difference occurs because the LU approach vectorizes the do loop associated with the number of right sides, and the SPP vectorizes in the direction of the tridiagonal solve. With some creative programming, one could potentially vectorize the entire SPP approach with a single array, while the LU approach can vectorize over the right-side arrays.

In the final experiment, the ability of SPP to control truncation error is demonstrated. The highest order of accuracy in the solution is based on the truncation error of the compact-difference

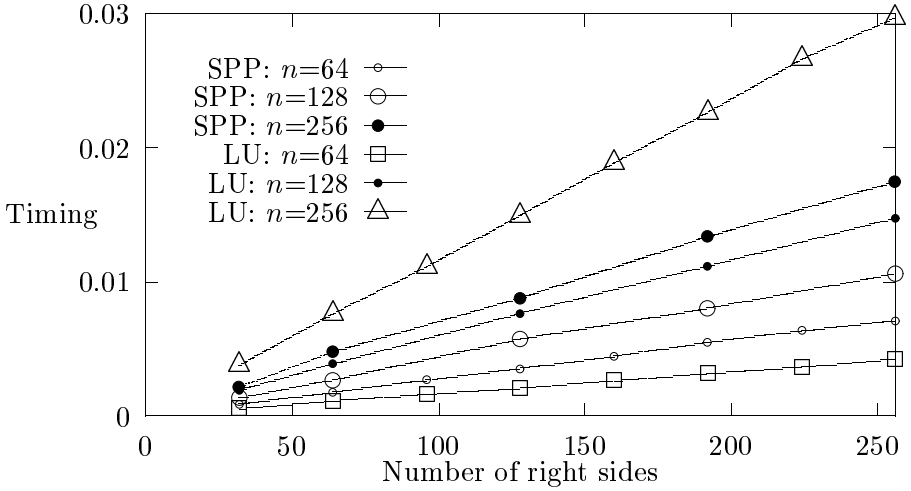

Figure 13. Timing of SPP and LU algorithms: multiple right sides.

approaches in equations (2) and (3). As a result, to require machine-zero is overkill for the compact solver and leads to unnecessary computational cost. By using the inequality (38), the choice of truncation can be determined based on a desired error bound. Figure 14 shows the SPP results of truncations $\bar{k} = 3$ and $\bar{k} = 5$, which correspond to errors 10^{-5} and 10^{-14}, respectively. If the accuracy of the SPP is relaxed, the computational cost decreases by a factor of 2.

6 Conclusion

A central goal of parallel processing is to achieve better, more accurate solutions. Because obtaining more accurate solutions, in general, means adding more discretization points, larger systems result and require greater computational power. The accuracy of a simulation solution is also bounded by the discretization scheme used. A clear requirement for obtaining a more accurate solution is to adopt discretization methods with high-order accuracy. Previously, a highly accurate discretization scheme, the *compact finite-difference scheme* [15], has been proposed. However, the almost symmetric Toeplitz tridiagonal systems that arise from compact schemes are sequential in nature and difficult to solve efficiently on parallel computers. In this paper, we have introduced a parallel algorithm, the *simple parallel prefix* (SPP) algorithm, for compact schemes and other related schemes.

The SPP algorithm is designed for fine-grain computing. With n processors, the SPP algorithm solves an n-dimensional system with $2\log(n) + 1$ AXPY operations. Two prefix communications are required in the solving phase and one broadcast communication is required in the modification

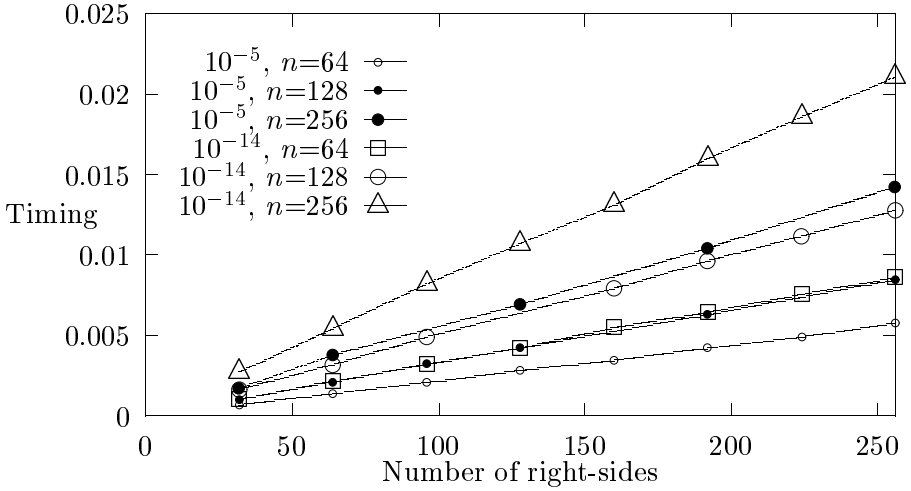

Figure 14. Timing of SPP with different accuracies.

phase. In comparison with existing tridiagonal solvers [19, 10], the SPP algorithm is simple in computing and simple in communication. It requires storage of only one $\log(n)$-dimensional vector for the computing phase and one n-dimensional vector for the modification phase. When the tridiagonal system is diagonally dominant, both the computing and the modification phases can be truncated without degrading the accuracy. Memory requirements will be further reduced when truncation is applied. A detailed accuracy analysis has been conducted to find the appropriate truncation number. Experimental results show that the SPP algorithm achieves a speedup greater than 1000 over the best sequential algorithm on a 16K PEs MasPar M-1 SIMD parallel computer. In addition to the good performance on the SIMD machines, the SPP algorithm also out performs the best sequential algorithm on a vector machine (Cray 2), even on systems with multiple right sides. Experimental and theoretical results show that the SPP algorithm is a good choice for compact schemes and for the emerging high-performance parallel computers.

The SPP algorithm is designed for symmetric and almost symmetric Toeplitz tridiagonal systems. It is a good candidate for compact schemes, alternating direction implicit method, wavelet collocation method, spline curve fitting, and many other scientific applications. It can be modified for different boundary conditions and for cases where the number of processors p is less than the dimension of the system. However, generalization of the algorithm for general tridiagonal systems or for band systems is unlikely.

The work presented in this paper is a continuation of efforts to design efficient parallel solvers for compact scheme. An efficient solver, the PDD algorithm, for coarse- or median-grain computing

has been proposed [21]. The PDD algorithm and the SPP algorithm can be combined on parallel machines with vector processing units.

Acknowledgements

We would like to thank J. Gustafson and M. Carter of the Scalable Computing Laboratory at Ames Laboratory for providing the access and general help on the Ames $16K$ PEs MarPar parallel computer.

References

[1] Y. Adams, *Highly accurate compact implicit methods and boundary conditions*, Journal of Computational Physics, 24 (1977), pp. 10–22.

[2] W. Cai and J. Wang, *Adaptive wavelet collocation methods for initial value boundary problems of nonlinear PDE's.* ICASE Technical Report, 93-48, ICASE, NASA Langley Research Center, 1993.

[3] M. H. Carpenter, D. Gottlieb, and S. Abarbanel, *The stability of numerical boundary treatments for compact high-order finite-difference schemes*, J. of Computational Physics, 108 (1993), pp. 272–295.

[4] K.-L. Chung and L.-J. Shen, *Vectorized algorithm for b-spline curve fitting on Cary X-MP EA/16se*, in Proc. of Supercomputing '92, 1992, pp. 166–169.

[5] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[6] O. Egecioglu, D. Koc, and A. Laub, *A recursive doubling algorithm for solution of tridiagonal systems on hypercube multiprocessors*, J. of Comp. and Appl. Math., 27 (1989).

[7] T. Eidson and G. Erlebacher, *Implementation of a fully-balanced periodic tridiagonal solver on a parallel distributed memory architecture*, Concurrency: Practice and Experience, 7 (1995).

[8] R. Hirsh, *Higher order accurate difference solutions of fluid mechanics problems by a compact differencing technique*, J. Comput. Phys., 19 (1975), pp. 90–109.

[9] C. Ho and S. Johnsson, *Optimizing tridiagonal solvers for alternating direction methods on boolean cube multiprocessors*, SIAM J. of Sci. and Stat. Computing, 11 (1990), pp. 563–592.

[10] R. Hockney, *A fast direct solution of Poisson's equation using Fourier analysis*, J. ACM, 12 (1965), pp. 95–113.

[11] R. Joslin, C. Streett, and C.-L. Chang, *Validation of three-dimensional incompressible spatial direct numerical simulation code.* NASA Technical Report, TP-3025, NASA Langley Research Center, July 1992.

[12] H. Kreiss and J. Oliger, *Methods for the approximate solution of time dependent problems.* GARP Report No 10, 1973.

[13] J. Lambiotte and R. Voigt, *The solution of tridiagonal linear systems on the CDC Star-100 computer*, ACM Trans. Math. Soft., 1 (1975), pp. 308–329.

[14] D. Lawrie and A. Sameh, *The computation and communication complexity of a parallel banded system solver*, ACM Trans. Math. Soft., 10 (1984), pp. 185–195.

[15] S. Lele, *Compact finite difference schemes with spectral-like resolution*, J. Comp. Phys., 103 (1992), pp. 16–42.

[16] M. A. Malcolm and J. Palmer, *A fast method for solving a class of tridiagonal linear systems*, Communications of the ACM, 17 (1974), pp. 14–17.

[17] J. Ortega and R. Voigt, *Solution of partial differential equations on vector and parallel computers*, SIAM Review, (1985), pp. 149–240.

[18] J. Sherman and W. Morrison, *Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix*, Ann. Math. Stat., 20 (1949).

[19] H. Stone, *An efficient parallel algorithm for the solution of a tridiagonal linear system of equations*, J. of ACM, 20 (1973), pp. 27–38.

[20] J. C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Wadsworth & Brooks/Cole, Mathematics Series, 1989.

[21] X.-H. Sun, *Application and accuracy of the parallel diagonal dominant algorithm.* ICASE Technical Report, 93-6, ICASE, NASA Langley Research Center, 1993. A short version appears in Proc. of ICPP '93.

[22] X.-H. Sun and J. Gustafson, *Toward a better parallel performance metric*, Parallel Computing, 17 (1991), pp. 1093–1109.

[23] X.-H. Sun and L. Ni, *A structured representation for parallel algorithm design on multicomputers*, in Proc. of the Sixth Conf. on Distributed Memory Computing, April 1991, pp. 596–599.

[24] ——, *Scalable problems and memory-bounded speedup*, J. of Parallel and Distributed Computing, 19 (1993), pp. 27–37.

[25] X.-H. Sun and D. Rover, *Scalability of parallel algorithm-machine combinations*, IEEE Transactions on Parallel and Distributed Systems, (1994), pp. 599–613.

[26] X.-H. Sun, H. Zhang, and L. Ni, *Efficient tridiagonal solvers on multicomputers*, IEEE Transactions on Computers, 41 (1992), pp. 286–296.

[27] H. Wang, *A parallel method for tridiagonal equations*, ACM Trans. Math. Software, 7 (1981), pp. 170–183.