CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2015; **27**:5076–5095
Published online 29 April 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3511

An iteration-based hybrid parallel algorithm for tridiagonal systems of equations on multi-core architectures

Guangping Tang[1], Wangdong Yang[1], Kenli Li[1,*,†], Yu Ye[1], Guoqing Xiao[1] and Keqin Li[1,2]

[1]*College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China*
[2]*Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

SUMMARY

An optimized parallel algorithm is proposed to solve the problem occurred in the process of complicated backward substitution of cyclic reduction during solving tridiagonal linear systems. Adopting a hybrid parallel model, this algorithm combines the cyclic reduction method and the partition method. This hybrid algorithm has simple backward substitution on parallel computers comparing with the cyclic reduction method. In this paper, the operation count and execution time are obtained to evaluate and make comparison for these methods. On the basis of results of these measured parameters, the hybrid algorithm using the hybrid approach with a multi-threading implementation achieves better efficiency than the other parallel methods, that is, the cyclic reduction and the partition methods. In particular, the approach involved in this paper has the least scalar operation count and the shortest execution time on a multi-core computer when the size of equations meets some dimension threshold. The hybrid parallel algorithm improves the performance of the cyclic reduction and partition methods by 19.2% and 13.2%, respectively. In addition, by comparing the single-iteration and multi-iteration hybrid parallel algorithms, it is found that increasing iteration steps of the cyclic reduction method does not affect the performance of the hybrid parallel algorithm very much.

Received 22 July 2014; Revised 16 March 2015; Accepted 18 March 2015

KEY WORDS: hybrid parallel algorithm; multi-core architecture; multi-threading; tridiagonal system of equations

1. INTRODUCTION

1.1. Background

A tridiagonal system of linear equations is represented by the form $Ax = d$, with possibly tridiagonal coefficient matrix A, a column vector of multiple lines x, and multiple right-hand sides d. Solving a tridiagonal system of linear equations is an integral and important part of many engineering applications and programs for scientific computation [1]. The applications of tridiagonal solvers include cubic spline approximations, alternating direction implicit method, and real-time or interactive applications in computer graphics, video games, animation films [2], and so on. With the recent development and availability of various parallel, vector, and super computers, sequential algorithms based on divide-and-conquer approaches were subsequently proposed, and these formed the foundations of a large number of researches on parallel tridiagonal solvers. Fast solvers become especially critical to running time performance for tridiagonal system. A better understanding of the performance is propelled by the need for scalable and fast parallel solvers of the tridiagonal equations.

*Correspondence to: Kenli Li, College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China.
†E-mail: lkl@hnu.edu.cn

In the past, several direct and iterative methods have been proposed for this problem. The iterative methods [3, 4] are mainly Jacobi's method, Gauss-Seidel, and successive overrelaxation. The direct methods include Thomas algorithm [5] and several parallel methods. Although the Thomas algorithm is the fastest algorithm on a serial computer, it is not directly and completely parallelizable because each step of this algorithm depends on the preceding one [6]. The present paper will describe several parallel tridiagonal solvers and present a new hybrid parallel algorithm for the tridiagonal systems.

1.2. Related research

The KLU (Clark Kent LU) algorithm, one method in existing numerical libraries, was proposed by Ekanathan Palamadai [7]. KLU is a sparse high-performance linear solver that employs hybrid ordering mechanisms and elegant factorization and solve algorithms. It achieves high quality fill-in rate and beats many existing solvers in run time, when used for matrices arising in circuit simulation. But KLU is a fast serial algorithm.

Research into parallelization strategies of tridiagonal solvers continues to be an active area of exploration. The Thomas algorithm, a specific Gaussian elimination method, is one of the first algorithms considered for tridiagonal linear system. This algorithm can be completed in two distinct steps, forward elimination and backward substitution. The first step consists of 'removing the coefficients under the diagonal, and then this leaves the new system with the coefficient matrix of two diagonals. The second step is to exploit backward sweep eliminating for the new system in order to calculate the solution.

Part of the Thomas algorithm can also be parallelized. T.M. Edison and G. Erlebacher [8] transformed a sequential Thomas algorithm into the parallel, chained, load-balanced algorithm. Lambiotte and Voigt [3] presented the cyclic reduction method on the CDC STAR-100 computer. The cyclic reduction method implemented on GPU [9] is also one divide-and-conquer algorithm [10, 11]. In this approach, all the odd-indexed unknowns are eliminated while the even-indexed ones are left. Then, we can get half of the previous system of equations but with the same tridiagonal structure [12]. Finally, the odd-indexed unknowns in the reduced tridiagonal systems are recursively eliminated until only a minimal number of equations that can be directly solved are reached. The results of this trivial system are then recursively back-substituted to realize the full solution of the tridiagonal systems [13].

Stone first reported the recursive doubling method [14, 15]. In this algorithm, an LU [16] decomposition of the tridiagonal matrix A is carried out so that $Ax = LUx = Lz = y$ where $Ux = z$. This algorithm consists of two primary steps. In a forward sweep, the system $Lz = y$ is solved followed by a backward sweep solving $Ux = z$ [13]. Parallel extensions of the recurrence relationships that are used in the forward and backward sweeps of the recursive doubling algorithm use the recursive doubling form, in which each recursion relationship is expressed in the light of two functions that are each half as complex [6]. In addition, Zhang *et al.* [2] reported the parallel cyclic reduction and hybrid algorithms containing hybrid parallel cyclic reduction and hybrid recursive doubling algorithms.

The parallel partition method, another example of a divide-and-conquer algorithm, was first developed by Wang [1]. With this algorithm, the tridiagonal matrix A is first partitioned into a suitable number of sub-matrices. Then we do eliminations simultaneously for each of the sub-matrices until the coefficient matrix A is diagonalized [10]. Rozloznik *et al.* [17] presented a symmetric tridiagonalization algorithm. This algorithm effectively exploits high-performance level-3 Basic Linear Algebra Subprograms routines and could be as efficient as symmetric block-diagonalization.

Besides tridiagonal solvers, block tridiagonal solvers based on cyclic reduction and parallel cyclic reduction are particularly amenable to efficient and scalable parallelization [13, 18]. The recursive doubling method is developed for parallel computers and supercomputers such as the Illiac IV [19] consisting of multi-processors. The cyclic reduction method, the partition method, and other parallel algorithms can also be implemented efficiently on parallel computers. Among these methods, the

DOI: 10.1002/cpe

cyclic reduction method is previously found to be the most effective on a vector computer [20], and the partition method is unlikely to be more efficient than the cyclic reduction unless the size of tridiagonal systems is much larger than the partition size and the thread count.

1.3. Contribution

In this work, we present a new parallel algorithm of solving the tridiagonal linear systems. We combine the cyclic reduction method and the partition method and propose a new hybrid method as the hybrid parallel algorithm. This new method simplifies the backward substitution of the cyclic reduction method, so it can realize better performance than other related algorithms. The detailed realization of this algorithm can be partitioned to three steps.

Step 1: The tridiagonal system of equations is reduced to half size of the previous one by one or more iterations of the cyclic reduction method.
Step 2: We can then solve the half one with the partition method until each even-indexed unknown is worked out.
Step 3: The third step is backward substitution that is to plug the resolved parameters into the primitive tridiagonal system of equations to work out the other unknowns by Thomas algorithm.

To further research the performance of this hybrid parallel algorithm, several added iterations of the cyclic reduction algorithm are performed in contrast to the single-iteration hybrid parallel algorithm. By computing the operation counts, the vector operation counts and scalar operation counts, we compare these parallel algorithms in this paper. Note that there is different efficiency for each method with different operating condition. On the multi-threading implementation [21] involved in this paper, the hybrid algorithm achieves better efficiency than the other parallel methods and has the least scalar operation count and the shortest execution time on a multi-core computer when the size of an equation meets some dimension threshold.

1.4. Paper organization

The rest of the paper is organized as follows. Section 2 discusses three different algorithms for tridiagonal solvers, that is, Thomas algorithm, the cyclic reduction, and partition methods. Section 3 presents the new hybrid parallel algorithm based on iterations of the cyclic reduction method. In Section 4, we compare these three parallel algorithms by contrasting the operation counts; followed by Section 5 in which the real experimental results are presented for the solver executed on a parallel machine with multi-core nodes. We conclude and predict the future work in Section 6.

2. THREE TRIDIAGONAL SOLVERS

A tridiagonal system of n linear equations of the form $Ax = d$ is described in Eq. (1):

$$Ax \equiv \begin{pmatrix} a_1 & b_1 & & & & \\ c_2 & a_2 & b_2 & & & \\ & c_3 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & & c_n & a_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = d, \tag{1}$$

where A is a tridiagonal coefficient matrix and d is a column vectors with n lines.

For solving such a system of equations, except some iterative methods such as Jacobi's method, Gauss-Seidel, and successive overrelaxation, we mainly consider Thomas algorithm and other parallel algorithms. Although the iterative methods are general algorithms for serial computers, they are inappropriate for parallel computers. These iterative methods usually cost more time and become

DOI: 10.1002/cpe

less efficient. Especially, the successive overrelaxation method relates to choose a relaxation factor determining the performance of this method, but it is difficult to select a good one. While using parallel algorithms, we can receive efficient results on a multi-core computers [23–25]. Next, we discuss several direct parallel methods to solve the tridiagonal linear system.

2.1. Thomas algorithm

The Thomas algorithm is a classic algorithm to solve the tridiagonal system and is a simplified form of Gaussian elimination in the tridiagonal system of equations case [26]. The tridiagonal system can be written as

$$c_i x_{i-1} + a_i x_i + b_i x_{i+1} = d_i, \quad i = 1, 2, \ldots, n,$$

where $c_1 = 0$ and $b_n = 0$. This algorithm consists of two phases [1], forward elimination for eliminating the coefficient c_i and backward-substitution elimination for producing the solution. In the first step, we eliminate the lower diagonal by

$$\begin{aligned} b_1^{'} &= \frac{b_1}{a_1},\ a_1 = 1, b_i^{'} = \frac{b_i}{a_i - b_{i-1}^{'} c_i},\ a_i = 1, & i &= 2, 3, \ldots, n-1, \\ d_1^{'} &= \frac{d_1}{a_1},\ d_i^{'} = \frac{d_i - d_{i-1}^{'} c_i}{a_i - b_{i-1}^{'} c_i}, & i &= 2, 3, \ldots, n. \end{aligned}$$

The second step is aimed at solving all unknowns from last to first by the following:

$$x_n = d_n^{'}/a_n,\ x_i = d_i^{'} - d_i^{'} x_{i+1}.$$

The algorithm is simple and easy to comprehend but inherently serial and takes $O(n)$ computation steps, because the subsequent calculation depends on the result of the immediately preceding calculation. Furthermore, this algorithm can be unstable when the denominator $a_i - d_{i-1}^{'} c_i$ is zero or numerically zero. So we must consider other conditions or methods attached to this algorithm. In the following subsections, we present two parallel algorithms for this tridiagonal system, the cyclic reduction and partition methods. The two algorithms can take fewer steps and accommodate more efficient implementation on parallel computers.

2.2. Cyclic reduction

The cyclic reduction algorithm was proposed by Roger Hockney [27] and is one of the first implemented algorithms on Graphic Processing Unit. Cyclic reduction is a divide-and-conquer algorithm and mainly has two phases, forward reduction and backward substitution. After a forward reduction, all the odd-indexed unknowns are eliminated while the even-indexed ones are left. Then, we can get half the size of the previous system of equations but with the same tridiagonal structure. The odd-indexed unknowns in the reduced system of equations are then recursively eliminated until a system of two unknowns is reached. The two equations can be directly solved. For backward substitution, we plug the previously solved values into the sub-matrix from last to first and obtain the solution of the rest unknowns in the equations.

Same as the Thomas algorithm, we consider a tridiagonal linear system $Ax = d$. This tridiagonal system can be written in the form $c_i x_{i-1} + a_i x_i + b_i x_{i+1} = d_i$ $(i = 1, 2, \ldots, n)$, defining $x_0 = x_{n+1} = 0$, $c_1 = b_n = 0$. We present three consecutive equations of the system as follows:

$$\begin{pmatrix} c_{2i-1} & a_{2i-1} & b_{2i-1} & & \\ & c_{2i} & a_{2i} & b_{2i} & \\ & & c_{2i+1} & a_{2i+1} & b_{2i+1} \end{pmatrix} x = \begin{pmatrix} d_{2i-1} \\ d_{2i} \\ d_{2i+1} \end{pmatrix}, \tag{2}$$

where $x = (x_{2i-2}, x_{2i-1}, x_{2i}, x_{2i+1}, x_{2i+2})^T$. We eliminate x_{2i-1} and x_{2i+1} by multiplying $(2i-1)$th equation with $c_{2i}(a_{2i-1})^{-1}$, $(2i+1)$th equation with $-b_{2i}(a_{2i+1})^{-1}$ and add these to $(2i)$th equation. After that, we obtain the result of the $(2i)$th equation:

DOI: 10.1002/cpe

$$\left(c'_{2i}\ 0\ a'_{2i}\ 0\ b'_{2i}\right) x = d'_{2i}. \tag{3}$$

After eliminating all odd-indexed unknowns x_{2i-1} for each i from 1 to $n/2$, the result is a new tridiagonal system of linear equations in Eq. (4):

$$A_1 X_1 \equiv \begin{pmatrix} a_2 & b_2 & & & & \\ c_4 & a_4 & b_4 & & & \\ & \ddots & \ddots & \ddots & & \\ & & c_{n-2} & a_{n-2} & b_{n-2} & \\ & & & c_n & a_n & \end{pmatrix} \begin{pmatrix} x_2 \\ x_4 \\ \vdots \\ x_{n-2} \\ x_n \end{pmatrix} = D_1. \tag{4}$$

For the sake of simplicity, we diagrammatize the solving process of this tridiagonal system. Figure 1 shows the communication pattern of the algorithm. This figure can be partitioned into two portions. The upper half portion of the algorithm is forward reduction, and the lower half portion is backward substitution. We describe the cyclic reduction with several steps as follows.

1. Every three adjacent equations, the middle equation of which is $(2i)$th equality, are regarded as a group. So the system can be separated into $n/2$ groups as shown in (2).
2. Repeat the operations of Step 1 until system (4) reduces to a minimal system containing only two equations. Solve the minimal equations from Step 2 by Thomas elimination.
3. From the bottom-up, put the two resolved parameters back into the previous system to solve other unknowns.

Based on the description of the cyclic reduction algorithm, the cyclic reduction procedure is presented in Algorithm 1. In this approach, the forward reduction can be executed more easily than backward substitution. We use a simple and direct method, Thomas algorithm, to solve the

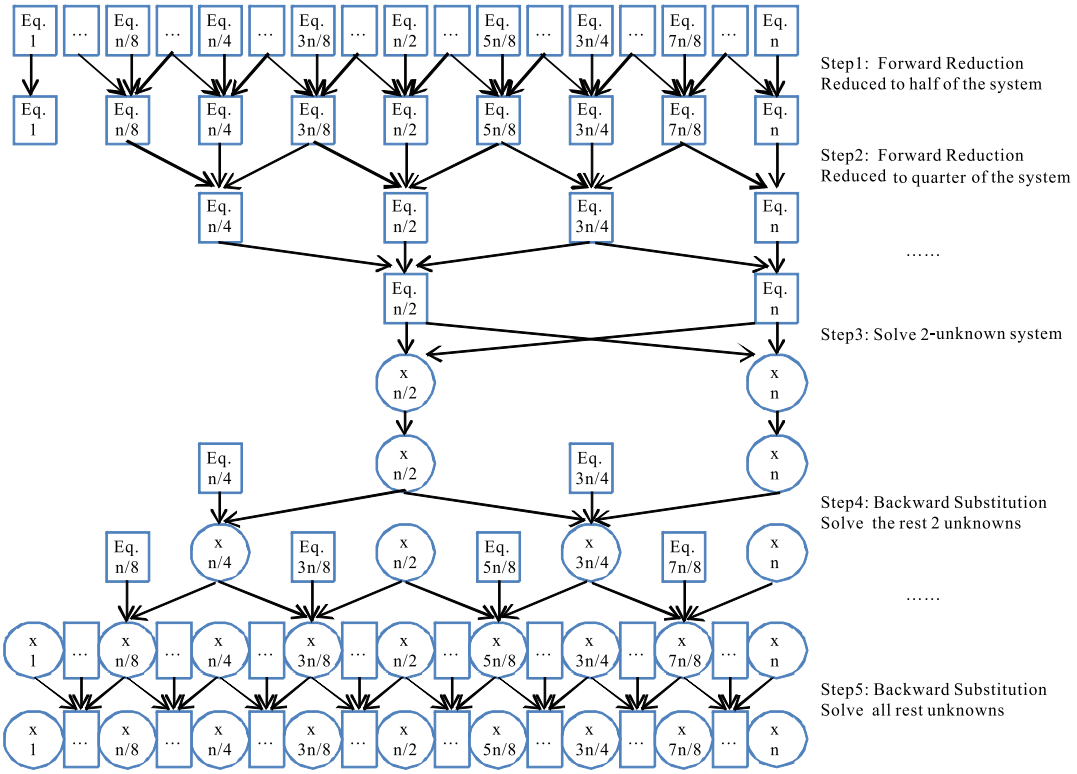

Figure 1. The solving process of the cyclic reduction for the tridiagonal system containing equations labeled Eq.1 to Eq.n. The segments in the boxes represent equations with unknowns, and the segments in the circles represent equations with the resolved variables.

DOI: 10.1002/cpe

Algorithm 1 : Cyclic Reduction Algorithm

Input:

A tridiagonal matrix A and a right-hand column

Output:

The solution of the tridiagonal systems and running time

Begin

1: Initialize the variables $x_0, x_{n+1} \leftarrow 0, c_1, b_n \leftarrow 0$
2: **for** $i = 1$ to $(n/2 - 1)$ simultaneously **do**
3: $e_{2i-1} \leftarrow c_{2i}/a_{2i-1}, e_{2i+1} \leftarrow b_{2i}/a_{2i+1}, c_{2i} \leftarrow (-c_{2i-1} * e_{2i-1}), b_{2i} \leftarrow (-b_{2i+1} * e_{2i+1}),$
$a_{2i} \leftarrow (a_{2i} - b_{2i-1} * e_{2i-1} - c_{2i+1} * e_{2i+1}), d_{2i} \leftarrow (d_{2i} - d_{2i-1} * e_{2i-1} - d_{2i+1} * e_{2i+1})$
4: **end for**
5: **if** $i = n/2$ **then**
6: $e_{2i-1} \leftarrow c_{2i}/a_{2i-1}, c_{2i} \leftarrow (-c_{2i-1} * e_{2i-1}), a_{2i} \leftarrow (a_{2i} - b_{2i-1} * e_{2i-1}),$
$d_{2i} \leftarrow (d_{2i} - d_{2i-1} * e_{2i-1})$
7: **end if**
8: Repeat the operations for the equations obtained in above step with cyclic reduction until the system reaches a system of two unknowns
9: Solve the two-unknown system
10: Backward substitution solves the rest two unknowns
11: Backward substitution solves the rest four unknowns
12: Following on, until work out all the rest unknowns
13: Return the solution X

End

unknowns in the process of backward substitution. On a parallel computer with multi-processors, the Thomas algorithm requires $O(n)$ steps while the cyclic reduction algorithm only requires $O(\log n)$ steps for the whole solving process.

2.3. Partition algorithm

The partition method was first proposed by Wang [1]. This algorithm is based on divide-and-conquer and is different from other parallel methods because it solves the tridiagonal system by block matrices. A tridiagonal system is first partitioned into block tridiagonal form, after which elimination can proceed simultaneously on all sub-matrices by elementary row transformations until finally the coefficients matrix A is diagonalized. The partition method roughly consists of three phases. We eliminate the coefficients under the main diagonal of main blocks, followed by the coefficients above the diagonal of main blocks. Then we eliminate the fill-ins created in the aforementioned process of elimination until the matrix A is diagonalized. During the process of elimination, we have included the right-hand-side computation (the process of computation for the right-side of the equations) along with the elimination.

The partition procedure is presented in Algorithm 2 in detail. There are mainly two critical issues with this technique that need to be mentioned. First, this algorithm is a parallel procedure, so there are different execution results when choosing diverse parallel patterns. Second, the selection of the partition size p is crucial to the operating efficiency. The partition method requires $O(\sqrt{n})$ steps on a parallel computer with n processors. We describe the four functions involved in the partition algorithm as follows.

ELIMC: It eliminates the elements c_s under the diagonal of main diagonal blocks from top to bottom and produces the new data set element values. During the elimination, it creates the nonzero elements f_s called 'fill-ins' in the (ik)th columns. In ELIMC$(ik + j)$, the subscripts of the matrix elements can be represented as the form of $ik + j$.

ELIMB: It eliminates the elements b_s above the diagonal of main diagonal blocks from the bottom-up and also produces the new data set element values. During the process of elimination, it also creates the nonzero elements g_s in the (ik)th columns. ELIMB$(ik + j - 1)$ and ELIMB(ik) represent the elimination way with the elements of different subscripts.

DOI: 10.1002/cpe

ELIMF: It eliminates the elements f_s under the diagonal of the new matrix from top to bottom. In ELIMF($ik + j$), the parallel operations occur within the blocks.

ELIMG: It eliminates the elements g_s above the diagonal of the new matrix from the bottom-up. Each block also performs the parallel operations solely in ELIMG($ik + j - 1$).

Algorithm 2 : Partition Method Algorithm

Input:

A tridiagonal matrix A and a right-hand column

Output:

The solution of the tridiagonal systems and running time

Begin

1: Partition the matrix A into a $p \times p$ block tridiagonal form and each block is a $k \times k$ tridiagonal matrix
2: Initialize d_0, $g_i (i = 0 \text{ to } p)$, $f_i (i = 1 \text{ to } k)$ and $f_{ik+1} (i = 0 \text{ to } p - 1)$ simultaneously
3: **for** $j = 2$ to k **do**
4: **for** each $i = 0$ to $p - 1$ simultaneously **do**
5: ELIMC($ik + j$), and produce the new data set element values f_s
6: **end for**
7: **end for**
8: **for** $j = k - 1$ to 2 **do**
9: **for** each $i = 0$ to $p - 1$ **do**
10: ELIMB($ik + j - 1$), and produce the new data set element values g_s
11: **end for**
12: **end for**
13: **for** each $i = 1$ to $p - 1$ simultaneously **do**
14: ELIMB(ik), and produce the new data set element values $g_k, g_{2k}, \ldots, g_{n-k}$
15: **end for**
16: **for** $i = 1$ to $p - 1$ **do**
17: **for** each $j = 1$ to k simultaneously **do**
18: ELIMF($ik + j$)
19: **end for**
20: **end for**
21: **for** $i = p - 1$ to 0 **do**
22: **for** each $j = 1$ to k simultaneously **do**
23: ELIMG($ik + j - 1$)
24: **end for**
25: **end for**
26: **for** each $i = 1$ to n **do**
27: $x_i = d_i / a_i$
28: **end for**
29: Return the solution X

End

3. A HYBRID PARALLEL ALGORITHM

We will use the cyclic reduction and partition methods as basic building blocks and propose a hybrid parallel algorithm, because the cyclic reduction has complicated backward substitution. The hybrid parallel algorithm actually contains two algorithms. We first use the cyclic reduction to reduce the tridiagonal system to a new tridiagonal systems. Next, we solve the reduced tridiagonal systems with the partition method. For the remaining systems, we finally solve these equations with Thomas algorithm. Figure 2 presents the execution process of the hybrid parallel algorithm and the associa-

DOI: 10.1002/cpe

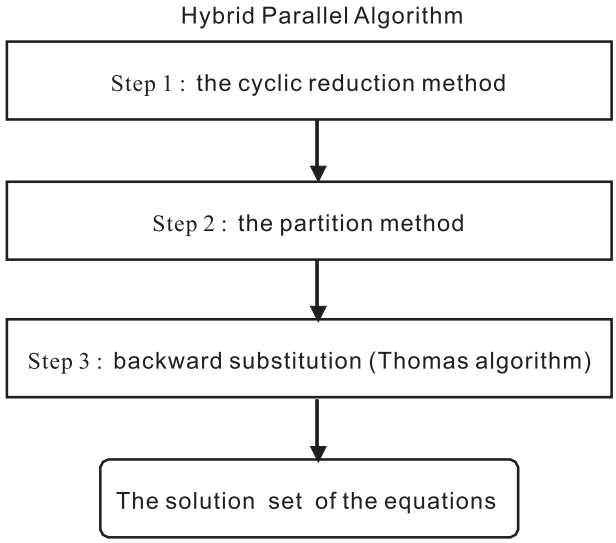

Figure 2. The three steps of the hybrid parallel algorithm.

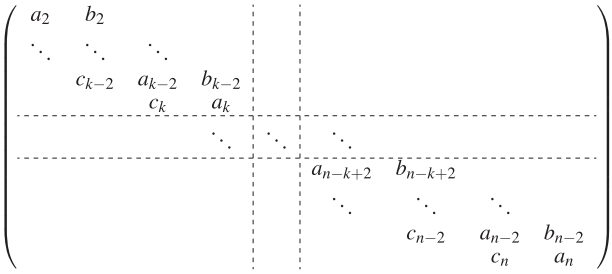

Figure 3. Tridiagonal coefficient matrix A_1 after elimination with the cyclic reduction method.

tion of three steps. The hybrid parallel algorithm overcomes the complicated backward substitution and makes the parallelism easier and more efficient. The following two sections will present the hybrid parallel algorithm-based iterations in detail.

3.1. Single-iteration hybrid parallel algorithm

In the forward reduction of the cyclic reduction algorithm, one iteration or multiple iterations can be chosen to execute. For one iteration of the cyclic reduction, the hybrid parallel algorithm can be outlined as follows.

1. Halve the system by the cyclic reduction method. As the description of the cyclic reduction, the system can be separated into $n/2$ groups as shown in (2). A new $(2i)$th equation can be obtained; by that, we eliminate the unknown x_{2i-1} of $(2i)$th equality using $(2i-1)$th and $(2i)$th equations, and the unknown x_{2i+1} of $(2i)$th equality using $(2i+1)$th and $(2i)$th equations as shown in (3). After eliminating the odd-indexed elements with the cyclic reduction method, it leaves us a new system with $n/2$ equations only containing the unknowns whose subscripts are even as shown in (4).
2. Solve the new half system left in Step 1 by partition algorithm. As described in partition algorithm, partition the matrix A_1 into a $p \times p$ block tridiagonal form and each block is a $k \times k$ matrix as shown in Figure 3. The right column D_1 is synchronously divided into p blocks, and each has k rows.
3. Eliminate the elements under the diagonal of main diagonal blocks. First eliminate $c_4, c_{k+4}, \ldots, c_{n-k+4}$ simultaneously, then eliminate $c_6, c_{k+6}, \ldots, c_{n-k+6}$ simultaneously,

DOI: 10.1002/cpe

and finally eliminate $c_k, c_{2k}, \ldots, c_n$ simultaneously. The matrix is now upper triangular except for (ik)th $(i = 1, 2, \ldots, p-1)$ columns that create the 'fill-ins' f_s.

4. Eliminate the elements above the diagonal of main diagonal blocks. First eliminate $b_{k-4}, b_{2k-4}, \ldots, b_{n-4}$ simultaneously, then $b_{k-6}, b_{2k-6}, \ldots, b_{n-6}$ simultaneously, and finally $b_2, b_{k+2}, \ldots, b_{n-k+2}$ followed by $b_k, b_{2k}, \ldots, b_{n-k}$ simultaneously. It leaves us with a diagonal matrix except for (ik)th $(i = 1, 2, \ldots, p-1)$ columns that create the g_s.
5. Eliminate the elements under the diagonal of the new matrix. The coefficient matrix can then be triangulated by the elimination of $f_{k+2}, \ldots, f_{2k-2}, f_{2k}$ simultaneously, followed by $f_{2k+2}, \ldots, f_{3k-2}, f_{3k}$, and finally $f_{n-k+2}, \ldots, f_{n-2}, f_n$ simultaneously. This process creates no 'fill-ins'. Then, we eliminate the elements under the diagonal in the new matrix. The coefficient matrix can be diagonalized by the elimination of $g_{n-2}, \ldots, g_{n-k+2}, g_{n-k}$ simultaneously and so on, followed by $g_{2k-2}, \ldots, g_{k+2}, g_k$, and finally $g_{k-2}, \ldots, g_4, g_2$ simultaneously. This process creates no 'fill-ins'.
6. Solve the even-indexed unknowns easily. The last step leaves us with a diagonal matrix, in which the main diagonal elements are nonzero and others are zeroes. In Figure 4, the underlined elements have been solved, and they can be used to solve other unknowns by combining the corresponding coefficients.
7. Put the resolved values $x_{2i}(i = 1, 2, \ldots, n/2)$ back into the system to solve other unknowns. Choosing all the equations $(2i-1)$th, we simplify the $n/2$ equations as shown in Figure 4 and get a new system in which the coefficient matrix only contains odd-indexed elements.
8. Solve the new diagonal system by Thomas algorithm.
9. Integrate the solved values that are reached in partition algorithm and Thomas algorithm and obtain the solution of the initial system.

Algorithm 3 presents the procedure of the single-iteration hybrid parallel algorithm. This algorithm requires $O(\sqrt{n})$ steps on a parallel computer with n processors and can achieve better efficiency. Similarly, the computation of parallel patterns and the selection of the partition size p are crucial to the operating efficiency because an appropriate p can largely shorten the operating steps and runtime in paralleling process. Based on the functions discussed in the partition algorithm, we describe other three main functions involved in the hybrid parallel algorithm as follows.

EXECUTE_1: It halves the tridiagonal system by the cyclic reduction method. We use Step 1 of the cyclic reduction to produce a new tridiagonal system of $n/2$ equations only containing the even index unknowns.

EXECUTE_2: It continues to further simplify the new tridiagonal system of $n/2$ equations by the partition method. This function mainly consists of four subfunctions from partition algorithm and eliminates the elements outside the leading diagonal until the coefficient matrix changes into a diagonal matrix.

EXECUTE_3: It solves all the remaining unknowns. After solving the unknowns of the new tridiagonal system, we can obtain the solution of the other half of unknowns by Thomas algorithm. The computational formula has the form

$$\begin{pmatrix}
x_1 & \underline{x_2} & x_3 & \underline{x_4} & x_5 & \cdots & \cdots & \underline{x_{n-2}} & x_{n-1} & \underline{x_n} \\
a_1 & b_1 & & & & & & & & \\
c_2 & a_2 & b_2 & & & & & & & \\
 & c_3 & a_3 & b_3 & & & & & & \\
 & & c_4 & a_4 & b_4 & & & & & \\
 & & & c_5 & a_5 & \ddots & & & & \\
 & & & & \ddots & \ddots & \ddots & & & \\
 & & & & & \ddots & \ddots & b_{n-3} & & \\
 & & & & & & c_{n-2} & a_{n-2} & b_{n-2} & \\
 & & & & & & & c_{n-1} & a_{n-1} & b_{n-1} \\
 & & & & & & & & c_n & a_n
\end{pmatrix}$$

Figure 4. Coefficient matrix after backward substitution with the partition method.

DOI: 10.1002/cpe

Algorithm 3 : Single-iteration Hybrid Parallel Algorithm

Input:

A tridiagonal matrix A and a right-hand column

Output:

The solution of the tridiagonal systems and running time

Begin

Initialize the variables $x_0, x_{n+1} \leftarrow 0,\ c_1, b_n \leftarrow 0$.
Assign the original a_i, b_i, c_i, d_i to four new arrays u_i, v_i, w_i, l_i
for $j = 1$ to $n/2$ simultaneously **do**
  EXECUTE_1(cyclic reduction, $2j$)
end for
Partition the matrix A_1 into $p \times p$ block tridiagonal form, and each block is a $k \times k$ tridiagonal matrix
Initialize d_0, $g_i(i = 0 \text{ to } p)$, $g_{ik-2}(i = 1 \text{ to } p)$, $f_{2i}(i = 1 \text{ to } k/2)$, and $f_{ik+2}(i = 0 \text{ to } p-1)$ simultaneously
Execute the elimination of the secondary diagonal elements in parallel
EXECUTE_2(partition algorithm, $ik + 2j$) {
  ELIMC($ik + 2j$)
  ELIMB($ik + 2j - 2$)
  ELIMB(ik)
  ELIMF($ik + 2j$)
  ELIMG($ik + 2j - 2$) }
for each $i = 1$ to n simultaneously **do**
  $x_i = d_i / a_i$
end for
Return the solution X_1
Backward substitution solves all the unknowns
for $i = 1$ to $n/2$ simultaneously **do**
  EXECUTE_3($2i - 1$)
end for
Return the solution X_2
Combine the solution X_1 and X_2, and return the whole solution X

End

$$x_{2i-1} = l_{2i-1} - w_{2i-1} * x_{2i-2} - v_{2i-1} * x_{2i} / u_{2i-1},$$
$$i = 1, 2, \ldots, n/2.$$

It is observed that this process of backward substitution is superior to the one of the cyclic reduction method.

3.2. Multi-iteration hybrid parallel algorithm

We have described the single-iteration hybrid parallel algorithm in detail. To verify whether the hybrid parallel algorithm is suitable for more cases, we add iterations of the cyclic reduction method before executing the partition method. For multiple iterations, the forward reduction of the cyclic reduction method must perform two or more steps. The problem size of the reduced tridiagonal systems, which will be solved by the partition method, becomes smaller accordingly. The difference between multi-iteration hybrid algorithm and single-iteration hybrid algorithm mainly is the number of times that the cyclic reduction is executed before executing the partition algorithm. The multi-iteration hybrid algorithm further decreases the tridiagonal system size by the cyclic reduction before the partition method executed and adds backward substitutions of the cyclic reduction algorithm after the partition method and Thomas algorithm.

Algorithm 4 presents the description for multi-iteration hybrid parallel algorithm. In this algorithm, the parallel operations of the cyclic reduction and partition methods are similar to

DOI: 10.1002/cpe

Algorithm 4 : Multi-iterations Hybrid Parallel Algorithm

Input:

A tridiagonal matrix A and a right-hand column

Output:

The solution of the tridiagonal systems and running time

Begin

1: Initialize and assign these related variables
2: Reduce the tridiagonal systems of equations to a new systems $A_{new}x_{new} = d_{new}$ with two or multiple iterations of the cyclic reduction method.
3: Simplify the new equations $A_{new}x_{new} = d_{new}$ by the partition method until this x_{new} can be solved.
4: Backward substitution solves the tridiagonal systems whose problem size is twice of $A_{new}x_{new} = d_{new}$ as size by Thomas algorithm.
5: Perform one or more backward substitutions of the cyclic reduction method until getting the solution of all unknowns.
6: Return the solution X of the tridiagonal systems.

End

Algorithm 3. This algorithm also requires $O(\sqrt{n})$ steps on a parallel computer with n processors. In the new systems $A_{new}x_{new} = d_{new}$, A_{new} is still a tridiagonal matrix. The dimension of the tridiagonal matrix A_{new} is $n/4$ after two iterations of the cyclic reduction. Accordingly, the dimension of the matrix A_{new} is $n/8$ after three iterations of the cyclic reduction, and this number will be $n/2^m$ after m iterations. The multi-iteration hybrid algorithm needs $m - 1$ backward substitutions of the cyclic reduction method when this algorithm has m iterations of the cyclic reduction method.

4. OPERATION COUNTS

The criterion measuring an algorithm contains many aspects. The operation counts, which are indispensable when measuring the performance of an algorithm, include vector operation counts for parallel vector processing and scalar operation counts for the total operation steps. When implementing an algorithm on a parallel computer with enough cores or an unlimited number of processors, the vector operation counts are the only condition to be concerned. However, on machines with M processors that are not enough for parallelization, we must take the scalar operation counts (the length of each vector operation) into account. When computing the vector operation counts, we can calculate through primarily two parts, left-side computation (the process of computation for the left side of the equations) and right-side computation (the process of computation for the right side of the equations), except for dividing operations within some algorithms. We now give arithmetic counts for both vector and scalar operations.

In following tables, we note M as the expression of multiplication, A as the expression of addition, D as the expression of division, and L as the expression of length. The size of the tridiagonal system is n, and this number meets the condition $n = 2^m$ where m is a positive integer. Table I presents the vector operation count of each step and the total number of vector operations for the cyclic reduction. Operation counts for right-side computation are listed separately. The total of vector operations for the cyclic reduction algorithm is $18 \log_2 n - 4$.

Table II presents the vector operation counts of the partition method. To implement the algorithm on a parallel computer, we should ensure that the number of processors is at least equal to max (p, k). The notes are same as that of Table I. We assume that data are stored according to their natural ordering. The coefficient matrix is partitioned into $p \times p$ blocks, and each block is $k \times k$ submatrix. These tridiagonal coefficients are stored in consecutive memory locations [28]. And all the elements of the operand vectors are spaced k-words apart in memory or in consecutive memory locations. Then we do the same elimination operations on the similar locations for each block. During the elimination, we need to operate both the left-side computations and right-side computations. From

DOI: 10.1002/cpe

Table I. Operation counts of the cyclic reduction method.

	Divide	Length	Left-side computation	Right-side computation
Forward reduction ($n = 2^m$)				
Step1	3	$(n/2)$-2	4M+2A	2M+2A
		1	3M+2A	2M+2A
(L = $n/2$)		1	2M+1A	1M+1A
Step2	3	$(n/2^2)$-2	4M+2A	2M+2A
		1	3M+2A	2M+2A
(L = $n/2^2$)		1	2M+1A	1M+1A
.....	.....	.....	.....	.....
Step($\log_2 n - 1$)	3	0		
		1	3M+2A	2M+2A
(L = 2)		1	2M+1A	1M+1A
Step($\log_2 n$)		1	1M+1A	1M+1A
		1		1M
(L = 2)		1		2M+1A
Backward Substitution (two unknowns solved)				
Step($m + 1$)		2		2M+1A
Step($m + 2$)		2		2M+1A
		2		3M+2A
Step($m + 3$)		2		2M+1A
		6		3M+2A
.....	.....	.....	.....	.....
Step($2m - 1$)		2		2M+1A
		$(n/2)$-2		3M+2A
Total of vector for forward reduction			(3D+4M+2A)m-3M-1A	(2M+2A)m+2M
Total of vector for backward substitution			0	(3M+2A)m-1M-1A
Total of vector operations for this algorithm			$18m - 4 = 18\log_2 n - 4$	

Table II. Operation counts of the partition method ($n = pk$).

	Divide	Length	Left-side computation	Right-side computation
Step1				
Step2	$k - 1$	p	$(k - 1)$(1M+1A)	$(k - 1)$(1M+1A)
		$p - 1$	$(k - 1)$M	
Step3	$k - 2$	p	$(k - 2)$M	$(k - 2)$(1M+1A)
	1	$p - 1$	kM+$(k - 1)$A	1M+1A
Step4	$p - 1$	k	$(p - 1)$(1M+1A)	$(p - 1)$(1M+1A)
Step5	$p - 1$	k		$(p - 1)$(1M+1A)
	1	$k - 1$		1M+1A
Step6	1	n		1M
Total of vector counts	$2k + 2p - 2$		$(4k + p - 5)$M +$(2k + p - 3)$A	$(2k + 2p - 2)$M +$(2k + 2p - 3)$A
Total number of vector for this method			$12k + 8p - 15$	

this table, the total number of vector operations for the partition algorithm is $2k + 8p - 15$, and the minimum of which is $19.6\sqrt{n} - 15$ when $k = \sqrt{2n/3} \cong 0.8\sqrt{n}$ and $p = 1.25\sqrt{n}$.

The vector operation counts and steps of the single-iteration hybrid parallel algorithm are presented in Table III. Considering the detailed description for operation counts of the cyclic reduction and partition methods, we leave out some detailed description for the single-iteration hybrid parallel algorithm operation counts in Table III. After the cyclic reduction method, we solve the reduced tridiagonal system by the partition method, then by Thomas algorithm for backward substitution.

DOI: 10.1002/cpe

Table III. Operation counts of the single-iteration hybrid parallel algorithm.

	Divide	Length	Left-side computation	Right-side computation
Cyclic reduction - first ($n = 2^m$)				
Step 1	3	$(n/2) - 2$	4M+2A	2M+2A
		1	3M+2A	2M+2A
(L = $n/2$)		1	2M+1A	1M+1A
Partition method - then ($pk = n/2$)				
Step 2	$p * k$			
Total of vector for partition method	$2k + 2p - 2$		$(4k + p - 5)$M+ $(2k + p - 3)$A	$(2k + 2p - 2)$M+ $(2k + 2p - 3)$A
Thomas algorithm - last (Backward substitution)				
Step 3		1		2M+1A
		$n/2 - 1$		3M+2A
Total of vector operation counts			$8k + 4p - 2$	$4k + 4p + 4$
Total of vector for this hybrid algorithm	$12k + 8p + 2$ (which is at a minimum when $k = \sqrt{n/3}$)			

Table IV. Vector operation counts ($n = kp, n/2 = k_1 p_1$).

Methods	Left-side computation	Right-side computation	Total of computation
Cyclic reduction	$9\log_2 n - 4$	$9\log_2 n$	$18\log_2 n - 4$
Partition method	$4p + 8k - 10$	$4p + 4k - 5$	$8p + 12k - 15$
Hybrid algorithm	$4p_1 + 8k_1 - 2$	$4p_1 + 4k_1 + 4$	$8p_1 + 12k_1 + 2$

Table V. Scalar operation counts ($n = kp, n/2 = k_1 p_1$).

Methods	Left-side computation	Right-side computation	Total of computation
Cyclic reduction	$9n - 10\log_2 n - 6$	$9n - 12\log_2 n$	$18n - 22\log_2 n - 6$
Partition method	$12n - 8p - 6k$	$9n - 4p - 2k - 4$	$21n - 12p - 8k - 4$
Hybrid algorithm	$9n - 8p_1 - 6k_1 - 1$	$6.5n - 4p_1 - 2k_1 - 6$	$15.5n - 12p_1 - 8k_1 - 7$

when using the partition method to compute the reduced system, one of the most remarkable things is $p \times k = n/2$ rather than $p \times k = n$. We assume that m is a positive integer and n is a power of 2. From this table, the total number of vector operation counts for the single-iteration hybrid parallel algorithm is $12k + 8p + 2$, and its least value is $8\sqrt{3n} + 2$ when $k = \sqrt{n/3} \cong 0.58\sqrt{n}$ and $p = \sqrt{3n}$.

Tables IV and V summarize the computations of these three algorithms. Table IV presents the total of vector operation counts of the algorithms, and Table V presents the scalar operation counts that are summations of operations in each step. On vector computers, both vector and scalar counts are important. When the vector lengths become large, the vector counts become less significant. In the tables, the left-side computations are obtained via the summation of left-side computations, respectively, and the corresponding length in the aforementioned three tables. From the formulas in the table, we can distill that the single-iteration hybrid parallel algorithm is most efficient as far as the total number of vector counts when n is small. If n is slightly larger, the total number of vector counts of the cyclic reduction algorithm is the smallest, and this algorithm may be the fastest regardless of the scalar counts. If considering the scalar counts, Table V shows that the single-iteration hybrid parallel algorithm is the most efficient than other two algorithms, and the partition method may be the worst when n is larger. With respect to computational complexity, although these three algorithms are $O(n)$, the single-iteration hybrid parallel algorithm can be

DOI: 10.1002/cpe

the fastest method because its coefficient is minimum. Figure 5 shows the comparison of the vector counts and scalar counts between three algorithms. When selecting different partition size p, the total number of vector counts or scalar counts for the partition algorithm and single-iteration hybrid parallel algorithm are not the same. Although based on the vector counts, the cyclic reduction seems to be the best, the scalar counts largely determine the superiority of algorithms in this work.

The aforementioned tables and Figure 5 present detailed description for the single-iteration hybrid parallel algorithm compared with two other parallel methods. The multi-iteration hybrid parallel algorithm is different from the single-iteration hybrid algorithm. Considering that the scalar counts are main influence factors for algorithms in the following work, we summarize only the scalar operation counts of this hybrid algorithm with different iterations in Table VI. With multiple iterations, the multi-iteration hybrid algorithm has less left-side computations, more right-side computations, and smaller total number of the scalar counts. If the cyclic reduction method makes t iterations, the total number of scalar counts for multi-iteration hybrid parallel algorithm can be computed by the formula

$$S_H(t) = 15n + \frac{n}{2^t} - 12p_t - 8k_t - 7t,$$

where $S_H(t)$ is the total number of scalar counts, n is the problem size of the tridiagonal systems, and $p_t k_t = n/2^t$.

Though multiple iterations of the cyclic reduction decrease the total scalar counts, the decreased range is narrow. It is obvious that this number of the total scalar counts is always about $15n$. The other factors we must consider are that the p_t and k_t become smaller when t becomes

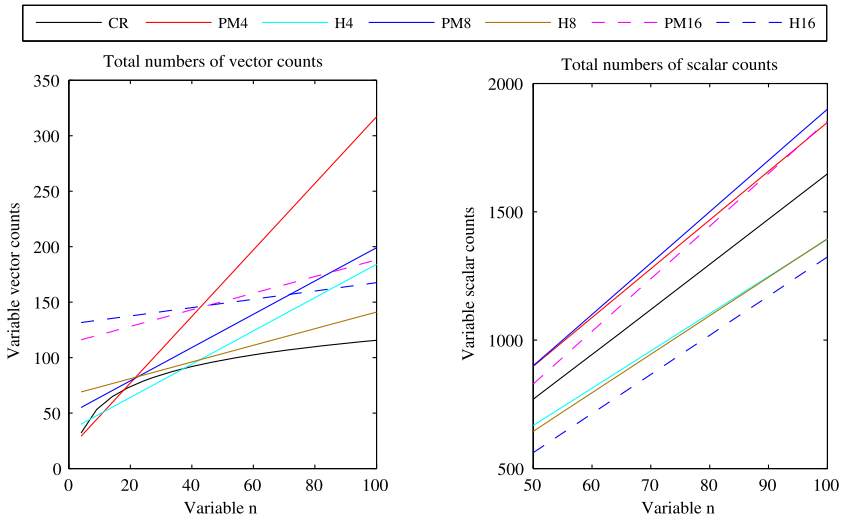

Figure 5. Comparison of three algorithms with vector and scalar operation counts. CR, cyclic reduction; PM4, partition method with $p = 4$; PM8, partition method with $p = 8$; PM16, partition method with $p = 16$; H4, the single-iteration hybrid parallel algorithm with $p = 4$; H8, the single-iteration hybrid parallel algorithm with $p = 8$; H16, the single-iteration hybrid parallel algorithm with $p = 16$.

Table VI. Scalar operation counts of multi-iteration hybrid algorithm ($n = kp, n/2^t = p_t k_t, t = 2, 3, 4, 5$).

Iterations	Left-side computation	Right-side computation	Total numbers of scalar counts
Two-iterations	$7.50n - 8p_2 - 6k_2 - 2$	$7.75n - 4p_2 - 2k_2 - 12$	$15.25n - 12p_2 - 8k_2 - 14$
Three-iterations	$6.75n - 8p_3 - 6k_3 - 3$	$8.38n - 4p_3 - 2k_3 - 18$	$15.13n - 12p_3 - 8k_3 - 21$
Four-iterations	$6.38n - 8p_4 - 6k_4 - 4$	$8.69n - 4p_4 - 2k_4 - 24$	$15.06n - 12p_4 - 8k_4 - 28$
Five-iterations	$6.19n - 8p_5 - 6k_5 - 5$	$8.84n - 4p_5 - 2k_5 - 30$	$15.03n - 12p_5 - 8k_5 - 35$

DOI: 10.1002/cpe

larger. Hence, an excess of iterations of the cyclic reduction has little impact on the efficiency of the multi-iteration hybrid parallel algorithm. For five iterations, the total scalar counts $S_H(t)$ is $15.03n - 12p_5 - 8k_5 - 35$, which is not more than $15.03n - 2\sqrt{3n} - 35$.

5. EXPERIMENTAL RESULTS

The following test environment has been used for all benchmarks. Most experiments are tested on a computer with two Intel Xeon Core E5506 CPUs (Santa Clara, CA, USA) running at 2.13 GHz and 8 GB of main memory. Each CPU has four cores. The test machine runs the Ubuntu 12.10 Linux 64 bit Operating System (London, UK). In order to show whether the hybrid parallel algorithm is generally more effective than the cyclic reduction and partition methods, we choose three different p values for testing. All the evaluation results are an average of 20 times under the same parameter settings.

OpenMP is designed to support portable implementation of parallel programs for shared memory multiprocessor architectures. It is a set of compiler directives and callable runtime library routines that extend C and C++ to express shared memory parallelism. It provides an incremental path for parallel conversion of any existing software, as well as targeting at scalability and performance for a complete rewrite or entirely new software. OpenMP is simpler than traditional Multi Point Interface (MPI) in writing parallel programs, because OpenMP uses multi-threading, which spends less time than MPI of using multi-process. Therefore, we implement these three parallel algorithms based on OpenMP.

Table VII shows the experimental data obtained through repeated experiments with an approximate average evaluation. These experiments are realized on the computer with eight processors used synchronously.

The KLU algorithm, a fast serial algorithm, has better computational efficiency than three parallel algorithms when the problem size is less than 2^{11}. However, the runtime of the KLU algorithm increases rapidly as doubling the tridiagonal system size continuously. The speedup of the parallel algorithms to the KLU algorithm increases to ten times or more when the size is 2^{15}, and this figure reaches more than 20 times when the size is 2^{19}. Communication scheduling of the parallel algorithms takes up part of the processing and elapsed time, and this scheduling time changes little as the problem size increasing. So the parallel algorithms take more time than the KLU algorithm when solving the smaller tridiagonal system, while take less time than the KLU algorithm when solving larger tridiagonal systems. Considering that three parallel algorithms have much better efficiency than the KLU algorithm, the following will only compare three parallel algorithms.

We can see that the single-iteration hybrid parallel algorithm is the fastest. The cyclic reduction costs more running time than the partition method when n is small, and less running time when n is larger than 2^{12}. From Table VII, several clear observations can be made. Although the single-iteration hybrid parallel algorithm is not always the optimal method in the process of the entire experiment, it can achieve better efficiency than the cyclic reduction and partition methods when the

Table VII. Runtime of three parallel algorithms and KLU algorithm in milliseconds.

n	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
Cyclic reduction	0.8556	0.9446	0.9552	1.1113	1.1971	1.4346
Partition method	0.8072	0.9262	0.9460	1.0052	1.2429	1.7158
Single-iteration	0.8814	0.9544	1.0815	1.1744	1.3235	1.4664
KLU algorithm	0.3556	0.4980	0.9204	1.8605	3.9901	8.1618
n	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
Cyclic reduction	1.9680	3.0720	5.1283	9.0811	17.2163	34.2585
Partition method	2.0243	2.7670	4.8543	8.7215	15.8901	31.9008
Single-iteration	1.7833	2.5507	4.5027	7.8140	14.4503	27.6950
KLU algorithm	16.561	32.793	66.515	140.27	302.806	612.439

DOI: 10.1002/cpe

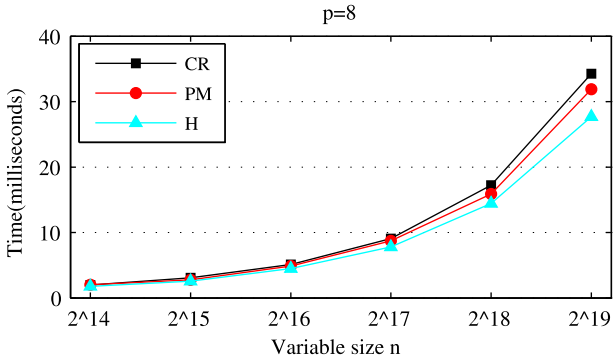

Figure 6. Performance comparison of three algorithms for different problem sizes with same partition size $p = 8$ on eight cores computer. H, the single-iteration hybrid parallel algorithm.

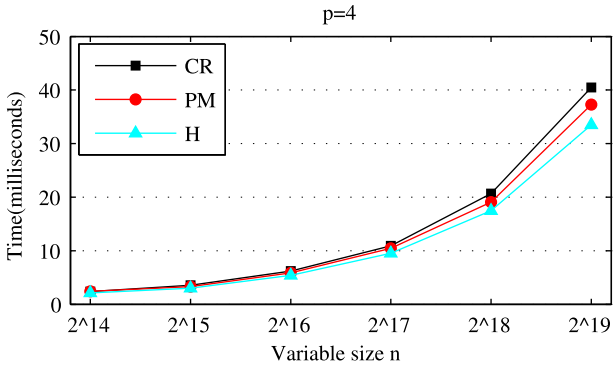

Figure 7. Performance comparison of three algorithms for different problem sizes with same partition size $p = 4$. Four threads are used to dispose the parallel computation synchronously.

size of the tridiagonal systems meets some dimension threshold. Especially, the larger the tridiagonal system size is, the more obvious the efficiency of the single-iteration hybrid parallel algorithm becomes. While the problem size is less than the 2^{12}, the execution time of three parallel algorithms is close for each system size, and the addition of the problem size usually only increases its execution time marginally.

In order to facilitate performance comparison of these algorithms, we select different p values and describe the performance by a scatter line chart for each different partition size p. Because the scale of changes is small when the problem size is less than 2^{13}, we choose the problem size above the 2^{13} to describe the runtime clearly on broken line graph. Figure 6 shows the performance comparison between three solvers of tridiagonal system with $p = 8$. The single-iteration hybrid parallel algorithm improves the performance of the cyclic reduction and partition methods by 19.2% and 13.2%, respectively. Figure 7 shows the performance comparison of these solvers for different problem sizes with $p = 4$. In the experiment, we use four threads synchronously to solve the parallel computation. From this figure, we can see that the runtime is generally more than one of Figure 6 for the same problem size. The hybrid algorithm also improves the performance of other two parallel methods by 17.2% and 12.5%, respectively. Figure 8 shows the comparison of three solvers with $p = 2$. In the process with double threads, the performance principle is the same as the four-thread solver. The runtime is accordingly more than one of Figure 7 with four threads. The performance of the cyclic reduction and partition methods are improved by 15.0% and 13.2%, respectively. So we find that when p changes from small to large for the same problem size, the execution time of the solvers decreases correspondingly. The execution time no longer shortens until an appropriate p reaches.

DOI: 10.1002/cpe

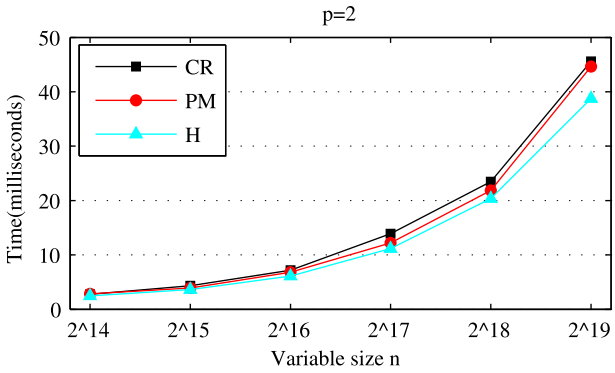

Figure 8. Performance comparison of three algorithms for different problem sizes with same partition size $p = 2$. Only two threads are used to dispose the parallel computation synchronously.

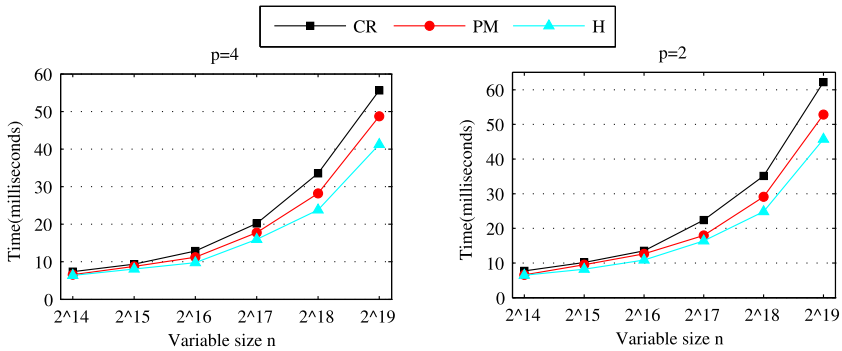

Figure 9. Performance comparison of three algorithms for different problem sizes with different partition size $p = 4$ or $p = 2$ on only four cores computer. That means four or two threads are used to dispose the parallel computation synchronously.

Figure 9 presents the performance results on other test platforms. This test computer is equipped with one AMD Athlon(tm) II 620 CPU (four cores) (Sunnyvale, CA, USA) running at 2.60 GHz and 4 GB memory. Its software environment is same as the preceding test computer. In the only four cores computer, four threads can be used to parallel computing synchronously. So we test these three parallel algorithms under two different partition sizes $p = 4$ and $p = 2$. The single-iteration hybrid parallel algorithm improves the performance of the cyclic reduction and partition methods by 20.1% and 15.4%, respectively, when $p = 4$. Accordingly, the values are improved by 22.5% and 14.7%, respectively, when $p = 2$. In Figures 7–9, the partition method has the shorter runtime than the cyclic reduction method. The single-iteration hybrid algorithm always executes the shortest runtime. From aforementioned figures, we can conclude that the single-iteration hybrid parallel algorithm is optimal than the cyclic reduction and partition methods.

To verify whether the hybrid parallel algorithm is suitable for more cases, we try multiple iterations of the cyclic reduction method under the same p value. We choose two to five iterations to experiment on two Intel Xeon Core CPUs. Also the different p values are used as factors assessing the multi-iteration of the cyclic reduction method. Figure 10 presents five different iterations of multi-iteration hybrid parallel algorithm for six diverse system sizes of the tridiagonal equations with $p = 8$. Each broken line represents one to five-iterations hybrid parallel algorithm under the same system size. The right box of this figure shows the tridiagonal system sizes that are the n power of two. Such as the problem size is $n = 2^{14}$ when exponent=14. In this figure, when the problem size is $n = 2^{19}$, the runtime of hybrid parallel algorithms is close to 27.7 milliseconds. Figure 11 also presents five different iterations for six diverse tridiagonal system sizes with $p = 4$. This indicates that only four threads are used synchronously on CPU. Figure 11 shows that runtime of hybrid parallel algorithms is close to 33.5 milliseconds while the problem size is $n = 2^{19}$. From

DOI: 10.1002/cpe

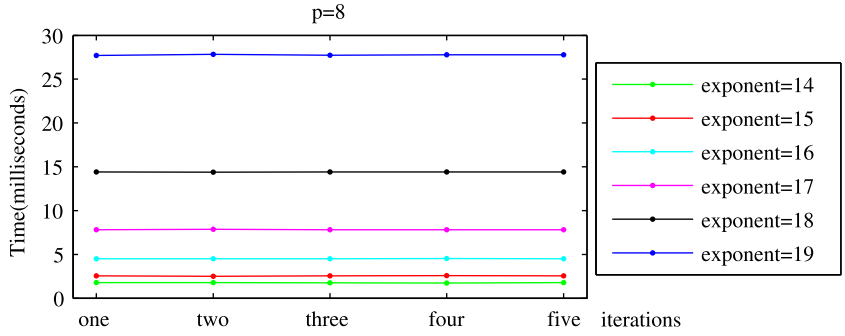

Figure 10. Runtime comparison of multi-iteration hybrid parallel algorithm with eight threads. The right section shows the tridiagonal system size, and 'exponent = 14' means that this size is 2^{14} and so on.

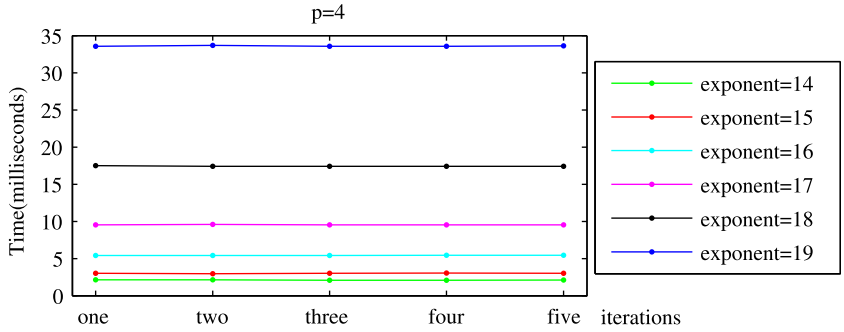

Figure 11. Runtime comparison of multi-iteration hybrid parallel algorithm with only four threads used synchronously. The right section shows the tridiagonal system size, and 'exponent = 14' means that its value is 2^{14} and so on.

Figures 10 and 11, we can easily observe two obvious characteristics. With the iteration step of the cyclic reduction method, the increased runtime is larger as the problem size increases. For same tridiagonal system size, the runtime of the hybrid parallel methods fluctuates within a small scope in the vicinity of a certain value. So the increase of iteration steps of the cyclic reduction method has little effect on the performance of the hybrid parallel algorithms.

So far, we have compared and analyzed all three tridiagonal solvers with multi-thread technology. By doing so, we have revealed the limitations and advantages of each algorithm. In order to further improve the performance, we use hybrid approach by combining the benefits of the cyclic reduction and partition methods. We also discover that the theoretical peak computing power is hard to reach because of various factors including not only divisions, bank conflicts, synchronizations, and time to access shared memory but also its internal pay, the load, the locality, and the pay of thread synchronization.

6. CONCLUSIONS AND FUTURE WORK

In this work, we mainly present four algorithms for solving the tridiagonal linear system. The direct methods considered in the paper are Thomas algorithm, the cyclic reduction, and partition methods. In particular, we have proposed the hybrid parallel algorithm that is suitable for both parallel and vector computers for solving a tridiagonal system of linear equations in this paper. Although the hybrid parallel algorithm may have slightly higher vector operation counts than the cyclic reduction, it has the least scalar operation counts. In this paper, we show how the hybrid solution achieves better performance with multi-threading programming technology. The main work is to study the execution times of these algorithms ignoring the computation error generated in the process of elimination and solving. From our several groups of experiments, we conclude that the hybrid parallel algorithm can be superior to the two competing algorithms. By comparing the single-iteration and

DOI: 10.1002/cpe

multi-iteration hybrid parallel algorithms, it is obvious that increasing iteration steps of the cyclic reduction method has little effect on the performance of the hybrid parallel algorithms. Restricted by the environment, we do not have parallel computers with plenty of cores to implement these parallel algorithms and cannot get higher efficiency of execution on a multi-core computers.

We plan to expand the study to include the effects of multi-threading parameters in the analysis as well as caching influences. The cost of transmitting a floating point number is assumed to be a constant in this analysis but actually is a variable in multi-core architectures. This study also proposes several directions for future research: (1) generalize the hybrid parallel algorithm for block tridiagonal matrices; (2) implement this hybrid algorithm on the platform of graphic processing unit with multi-core CPU; and (3) further improve the efficiency of execution on other parallel computers and supercomputers.

ACKNOWLEDGEMENTS

The authors are grateful to the two anonymous reviewers for their comments and suggestion to improve the manuscript. Thanks are also due to Qinghong Yang and Yulu Pan for the discussions on efficient multi-threading implementation of the parallel algorithms. The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005) and the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124).

REFERENCES

1. Wang HH. A parallel method for tridiagonal equations. *ACM Transactions on Mathematical Software* 1981; **7**: 170–183.
2. Zhang Y, Cohen J, Owens JD. Fast tridiagonal solvers on the GPU. *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP' 10)*, Bangalore, India, 2010; 127–136.
3. Lambiotte JJ, Jr., Voigt RG. The solution of tridiagonal linear systems on the CDC STAR-100 computer. *ACM Transactions on Mathematical Software* 1975; **1**:308–329.
4. Nguyen H, Beauregard MA, Morgan R. Improving the speed of convergence of GMRES for certain perturbed tridiagonal systems. *IEEE 45th Southeastern Symposium on System Theory(SSST)*, Waco, Texas, 2013; 63–67.
5. Thomas LH. Elliptic problems in linear difference equations over a network. Watson Scientific Computing Laboratory Report, Columbia University: Manhattan, 1949.
6. Hirshman SP, Perumalla KS, Lynch VE, Sanchez R. BCYCLIC: a parallel block tridiagonal matrix cyclic solver. *Journal of Computational Physics* 2010; **229**:6392–6404.
7. Natarajan EP. Klu-a high performance sparse linear solver for circuit simulation problems. Diss, University of Florida: Gainesville, 2005.
8. Edison TM, Erlebacher G. Implementation of a fully-balanced periodic tridiagonal solver on a parallel distributed memory architecture. *Concurrency and Computation: Practice and Experience* 1995; **7**(4):273–302.
9. Goddeke D, Strzodka R. Cyclic reduction tridiagonal solvers on gpus applied to mixed precision multigrid. *IEEE Transactions on Parallel and Distributed Systems* 2011; **22**:22–32.
10. Coakley ES, Rokhlin V. A fast divide-and-conquer algorithm for computing the spectra of real symmetric tridiagonal matrices. *Applied and Computational Harmonic Analysis* 2013; **34**:379–414.
11. Bini DA, Meini B. The cyclic reduction algorithm: from Poisson equation to stochastic processes and beyond. *Numerical Algorithms* 2009; **51**:23–60.
12. Sogabe T, El-Mikkawy M. Fast block diagonalization of k-tridiagonal matrices. *Applied Mathematics and Computation* 2011; **218**:2740–2743.
13. Seal SK, Perumalla KS, Hirshman SP. Revisiting parallel cyclic reduction and parallel prefix-based algorithms for block tridiagonal systems of equations. *Journal of Parallel and Distributed Computing* 2013; **73**:273–280.
14. Stone HS. Parallel tridiagonal equation solvers. *ACM Transactions on Mathematical Software* 1975; **1**:289–307.
15. Stone HS. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM* 1973; **20**:27–38.
16. Wu C-Y, Huang T-Z. Stability of block LU factorization for block tridiagonal block H-matrices. *Journal of Computational and Applied Mathematics* 2012; **236**:2673–2684.
17. Miro R, Shklarski G, Toledo S. Partitioned triangular tridiagonalization. *ACM Transactions on Mathematical Software* 2011; **37**:38–38:16.
18. Heller D. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM Journal on Numerical Analysis* 1976; **13**:484–496.
19. Barnes GH, Brown RM, Kato M, Kuck DJ, Slotnick DL, Stokes RA. The ILLIAC IV computer. *IEEE Transactions on Computers* 1968; **17**:746–757.
20. Andrew V. Terekhov: a fast parallel algorithm for solving block-tridiagonal systems of linear equations including the domain decomposition method. *Parallel Computation* 2013; **39**:245–258.

DOI: 10.1002/cpe

21. Akimova EN, Belousov DV. Parallel algorithms for solving linear systems with block-tridiagonal matrices on multi-core CPU with GPU. *Journal of Computational Science* 2012; **3**:445–449.
22. Petschow M, Bientinesi P. MR3-SMP: a symmetric tridiagonal eigensolver for multicore architectures. *Parallel Computation* 2011; **37**:795–805.
23. Tang G, Li K, Li K, Chen H, Jiayi D. A hybrid parallel tridiagonal solver on multi-core architectures. *2014 IEEE International Parallel Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, Phoenix (Arizona), USA, 2014; 604–613.
24. Ltaief H, Luszczek P, Dongarra J. High-performance bidiagonal reduction using tile algorithms on homogeneous multicore architectures. *ACM Transactions on Mathematical Software* 2013; **39**:16–16:22.
25. Li K, Yang W, Li K. Performance analysis and optimization for SpMV on GPU using probabilistic modeling. *IEEE Transactions on Parallel and Distributed Systems* 2015; **26**:196–205.
26. Heng Z, Wu Z. A parallel multi-layer hybrid algorithm for solving block-tridiagonal systems. *International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE)*, Changchun, China, 2011; 1415–1418.
27. Hockney Roger W. A fast direct solution of Poissons equation using Fourier analysis. *Journal of the ACM* 1965; **12**:95–113.
28. Allmann S, Rauber T, Runger G. Cyclic reduction on distributed shared memory machines. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference*, Mantova, Italy, 2001; 290–297.

DOI: 10.1002/cpe