Journal of Computational Physics 468 (2022) 111443

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Letter to the editor

Scalable parallel linear solver for compact banded systems on heterogeneous architectures

Hang Song[a,*], Kristen V. Matsuno[a], Jacob R. West[a], Akshay Subramaniam[b], Aditya S. Ghate[b], Sanjiva K. Lele[a,b]

[a] *Department of Mechanical Engineering, Stanford University, Stanford, CA 94305, USA*
[b] *Department of Aeronautics & Astronautics, Stanford University, Stanford, CA 94305, USA*

ARTICLE INFO

Article history:
Received 3 February 2021
Received in revised form 17 February 2022
Accepted 28 June 2022
Available online 19 July 2022

Keywords:
Compact banded system
Periodic boundary
Parallel cyclic reduction
Distributed memory
Parallel computing

ABSTRACT

A scalable algorithm for solving compact banded linear systems on distributed memory architectures is presented. The proposed method factorizes the original system into two levels of memory hierarchies, and solves it using parallel cyclic reduction on both distributed and shared memory. This method has a lower communication footprint across distributed memory partitions compared to conventional algorithms involving data transposes or re-partitioning. The algorithm developed in this work is generalized to cyclic compact banded systems with flexible data decompositions. For cyclic compact banded systems, the method is a direct solver with a deterministic operation and communication counts depending on the matrix size, its bandwidth, and the partition strategy. The implementation and runtime configuration details are discussed for performance optimization. Scalability is demonstrated on the linear solver as well as on a representative fluid mechanics application problem, in which the dominant computational cost is solving the cyclic tridiagonal linear systems of compact numerical schemes on a 3D periodic domain. The algorithm is particularly useful for solving the linear systems arising from the application of compact finite difference operators to a wide range of partial differential equation problems, such as but not limited to the numerical simulations of compressible turbulent flows, aeroacoustics, elastic–plastic wave propagation, and electromagnetics. It alleviates obstacles to their use on modern high performance computing hardware, where memory and computational power are distributed across nodes with multi-threaded processing units.

1. Introduction

In the past few decades, the use of graphics processing units (GPUs) in scientific computing has emerged as an attractive option to significantly accelerate various algorithms. The transition of several leadership class computing platforms to such heterogeneous architectures underscores the importance of numerical methods which can take full advantage of these nodes' parallel nature. The methods for solving certain linear systems presented in this work are well-suited for not only GPUs, but also platforms with hybrid memory management, and can take advantage of systems with distributed memory combined with multithreading.

[*] Corresponding author.
E-mail address: songhang@stanford.edu (H. Song).

https://doi.org/10.1016/j.jcp.2022.111443
0021-9991/© 2022 Elsevier Inc. All rights reserved.

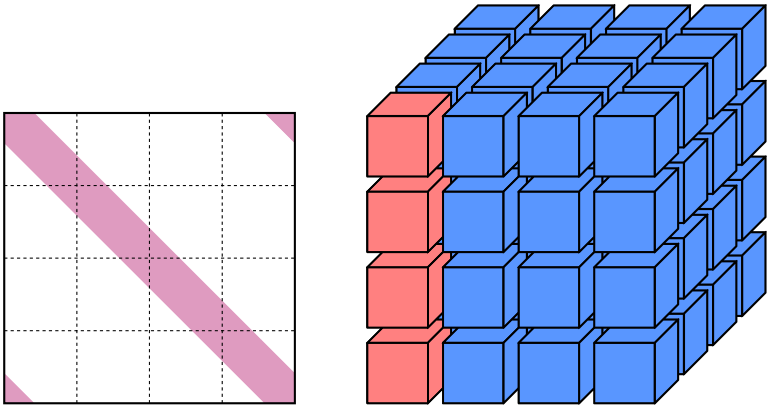

Fig. 1. Structure of cyclic banded linear system (left) and 3D grid decomposition (right). A pencil of chunks aligned in the solve direction forms a sub-group. Each sub-group constructs an individual linear system as shown on the left. An example sub-group is highlighted in red in the grid decomposition on the right. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

In multiscale physics problems, such as simulations of compressible turbulent flow, the resolution of both large and small scales on a discrete grid is essential. Similarly, computational applications involving hydrodynamic instabilities and wave propagation, such as in aeroacoustics, solid mechanics, and electromagnetics, require numerical discretizations with very low dispersion and dissipation errors. High order numerical methods have become increasingly attractive to tackle such problems since they provide high solution fidelity at a manageable computational cost [1]. Differentiation using compact finite difference schemes and elliptic solves using spectral methods can be represented discretely as compact banded systems, and are prime candidates for such multiscale computations due to their increased performance in the high wavenumber regime [2,3]. The desirable performance of compact schemes for resolving large ranges of scales has been demonstrated in incompressible [4–7] and compressible [8–11] turbulent flows, aeroacoustics [12,13] as well as multiphysics applications with complex physical phenomena [14,15]. These higher order finite differences are computed as a linear system with tridiagonal or other compact banded matrices. As derived by Lele [2], the tridiagonal schemes for collocated first derivatives, f', at gridpoint i with spacing $h = x_i - x_{i-1}$ are formulated as

$$\alpha f'_{i-1} + f'_i + \alpha f'_{i+1} = a\frac{f_{i+1} - f_{i-1}}{2h} + b\frac{f_{i+2} - f_{i-2}}{4h} \tag{1}$$

Similarly, interpolation between values on collocated and staggered grids can also be formulated as a tridiagonal system, where f is the original field and f^I is the interpolated field [16]:

$$\hat{\alpha} f^I_{i-1} + f^I_i + \hat{\alpha} f^I_{i+1} = a\frac{f_{i+1/2} + f_{i-1/2}}{2} + b\frac{f_{i+3/2} + f_{i-3/2}}{2} \tag{2}$$

For strong shock-turbulence interaction problems, the compact shock capturing schemes combined with a Riemann solver have been proved to be both robust and less dissipative [17,18]. For such schemes, block tridiagonal (or wider banded) systems will be formed, and the systems commonly remain well-conditioned as their size grows.

Multiphysics solvers for structured, Eulerian grids in a multidimensional domain may be decomposed as shown in Fig. 1, with each processor given access to a single chunk of the global domain. This decomposition is particularly useful for fixed, structured, Cartesian grids since the grid chunks on each processor can easily be determined from the decomposition layout using simple algebra. This method of grid decomposition facilitates workload distribution, and works particularly well for architectures with a distributed memory layout. Operations such as derivatives or interpolation along one dimension involve communication across a single row or column of grid partitioning, such as the chunks highlighted in red in Fig. 1. As shown in dotted lines in the matrix, sections of the matrix are initially distributed among several processors or nodes. Traditional parallel linear solvers of this system commonly assume uniform memory access. The solving process relies heavily on its communication requirements for data transpose. This work presents a linear solver for compact banded systems with highly scalable properties. First, a brief review of cyclic reduction (CR) and parallel cyclic reduction (PCR) for banded matrices is given. Section 2 illustrates the generalized PCR for generic acyclic compact banded systems, which serves as a building block of the proposed algorithm. Section 3 describes in detail the solution process for tridiagonal matrices of arbitrary size on an arbitrary number of processors, followed by an analytical extension of the method for other compact banded matrices. Section 4 provides additional implementation details to improve performance. In Section 5, a demonstration is provided of the computational performance of the linear solver and its use to solve the Navier-Stokes equations for the Taylor-Green vortex problem.

CR is a popular direct solve algorithm for structured matrix linear systems, particularly block tridiagonal linear systems [19]. It recursively reduces a linear system to half-size sub-systems until the size of the sub-system (typically 1 × 1) makes

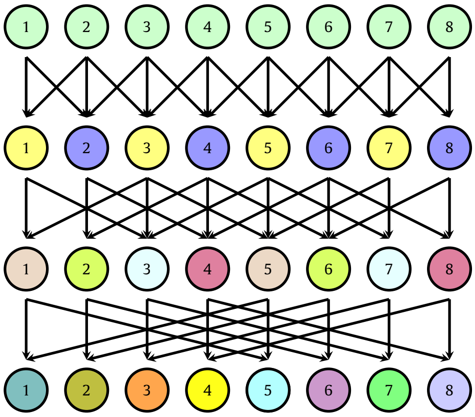

Fig. 2. Communication pattern of PCR for an 8 × 8 non-cyclic tridiagonal system. The sub-systems in each step are grouped by the same colors. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

it affordable to solve. Once the sub-system is solved, the result can propagate backward to the parent system to solve for the remaining unknowns. Hockney [20] initially derived CR in combination with the fast Fourier transform as an alternative algorithm for iterative solvers for the Poisson equation. Later, Buzbee et al. [21] presented a unified formulation and generalization of Hockney's CR and Buneman's [22] algorithm, which had mathematically equivalent reduction processes but differences in round-off errors and stability. Sweet [23,24] further generalized CR from matrices with block sizes of power-of-two to matrices of arbitrary block sizes. Similarly, Swarztrauber [25] also generalized CR for tridiagonal systems associated with separable elliptic equations. A parallel variant of CR, also known as PCR, was introduced by Hockney and Jesshope [26]. In the PCR process, the upper and lower off-diagonal elements of both the even and odd indexed rows of a tridiagonal matrix are simultaneously eliminated by the previous and the next rows in one step of reduction. As a consequence, it splits a system into two half-size sub-systems in each step of PCR. The communication pattern of an 8 × 8 non-cyclic tridiagonal system is shown in Fig. 2. After enough recursive splitting, all the sub-systems are of effectively trivial size, e.g. 1 × 1 in the bottom layer of Fig. 2, to solve all the unknowns in parallel. This means that PCR solves the linear system in a single forward pass and does not require a backward substitution phase.

Recent works have optimized both CR and PCR for modern parallel computer architectures, and have achieved considerable performance improvements for specific applications. For example, a GPU implementation is suggested by Zhang et al. [27], and the works of Hirshman et al. [28] and Seal et al. [29] improve the algorithm for block tridiagonal systems with large dense blocks. Nevertheless, most of the general PCR solvers are implemented for shared memory data access, and few improved algorithms have comprehensively considered data partitioning for distributed memory. The parallel linear solver developed in this paper is based on the concept of PCR to solve the banded system, and optimized for the grid decomposition on the distributed memory shown in Fig. 1. These banded systems typically are (block) tridiagonal or (block) pentadiagonal systems, but the present algorithm can be extended to wider bandwidths.

2. Generalized parallel cyclic reduction method

Beyond the tridiagonal system, PCR can be easily generalized for a compact banded system with arbitrary bandwidth. In order to form two sub-systems grouped by the even and odd rows, each row in the parent system, after a reduction step, is staggered with a zero entry between any of the two non-zero entries on the diagonal and off-diagonals, as shown in Fig. 3. In the generalized PCR approach, the total number of neighboring rows involved to eliminate the entries in row i equals the number of the off-diagonal elements. And the resulting row i is the linear combination of row i and the neighboring rows.

Let $\boldsymbol{a}_i^T$ be the i-th row vector in the parent matrix, and the reduction operation to obtain the i-th row vector in the resulting matrix, $\check{\boldsymbol{a}}_i^T$, can be expressed as

$$\check{\boldsymbol{a}}_i^T = \boldsymbol{a}_i^T - \sum_{j=1}^{(w-1)/2} \left(k_{+j} \boldsymbol{a}_{i+j}^T + k_{-j} \boldsymbol{a}_{i-j}^T \right) \tag{3}$$

where w is the bandwidth of the compact banded system. During a reduction step, each of the zero staggered entries can be formed with a unique linear combination of the involved neighboring row vectors, as the boxed columns in Fig. 3. The coefficients, k_{+j} and k_{-j}, can be solved from the linear system described in Equation (4).

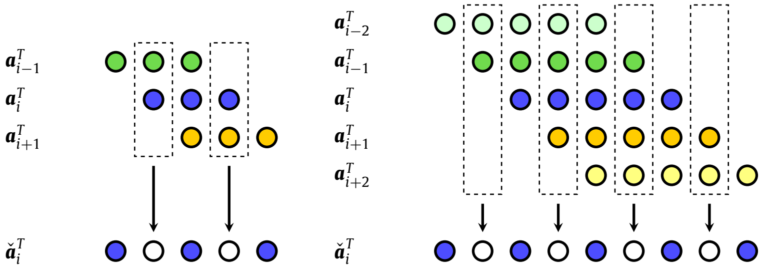

Fig. 3. Example of one step in generalized PCR: (a) tridiagonal system; (b) penta-diagonal system. The colored circles are non-identically-zero entries and the uncolored circles are identically-zero entries. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$\begin{bmatrix} \ddots & \ddots & \ddots & & & \\ \cdots & a_{i-2,i-3} & a_{i-1,i-3} & \cdots & & \\ \cdots & a_{i-2,i-1} & a_{i-1,i-1} & a_{i+1,i-1} & \cdots & \\ & \cdots & a_{i-1,i+1} & a_{i+1,i+1} & a_{i+2,i+1} & \cdots \\ & & \cdots & a_{i+1,i+3} & a_{i+2,i+3} & \cdots \\ & & & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ k_{-2} \\ k_{-1} \\ k_{+1} \\ k_{+2} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ a_{i,i-3} \\ a_{i,i-1} \\ a_{i,i+1} \\ a_{i,i+3} \\ \vdots \end{bmatrix} \tag{4}$$

Specifically, for a tridiagonal parent system ($w = 3$), k_{+j} and k_{-j} for each row i are governed by a 2×2 diagonal system shown in Equation (5). For a penta-diagonal parent system ($w = 5$), $a_{i,i-3} = a_{i,i+3} = 0$ for each row i, and k_{+j} and k_{-j} are governed by a 4×4 tridiagonal system described in Equation (6).

$$\begin{bmatrix} a_{i-1,i-1} & \\ & a_{i+1,i+1} \end{bmatrix} \begin{bmatrix} k_{-1} \\ k_{+1} \end{bmatrix} = \begin{bmatrix} a_{i,i-1} \\ a_{i,i+1} \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} a_{i-2,i-3} & a_{i-1,i-3} & & \\ a_{i-2,i-1} & a_{i-1,i-1} & a_{i+1,i-1} & \\ & a_{i-1,i+1} & a_{i+1,i+1} & a_{i+2,i+1} \\ & & a_{i+1,i+3} & a_{i+2,i+3} \end{bmatrix} \begin{bmatrix} k_{-2} \\ k_{-1} \\ k_{+1} \\ k_{+2} \end{bmatrix} = \begin{bmatrix} 0 \\ a_{i,i-1} \\ a_{i,i+1} \\ 0 \end{bmatrix} \tag{6}$$

3. Parallel linear solver for compact banded system

This section will introduce the parallel direct solver used for solving compact banded linear systems with the data partition on the distributed memory. Consistent with the grid decomposition pattern in Fig. 1, the compact banded linear system, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, is also correspondingly decomposed into a sparse block tridiagonal system [30] shown in Fig. 4. The data in $\boldsymbol{x}$ and $\boldsymbol{b}$ are stored in the distributed memory. Typically, applications require solving multiple systems with the same matrix $\boldsymbol{A}$, like in Fig. 1. Then, $\boldsymbol{x}$ and $\boldsymbol{b}$ can be a batch of vectors forming an $N \times M$ matrix where M is the number of independent solutions needed. The subscripts in Fig. 4 indicate the rank of the aligned grid decomposition. Each rank has access to the data stored in its shared memory, the boundaries of which are indicated by dotted lines. $\widetilde{\boldsymbol{D}}_i$ is an $r \times r$ dense square matrix, whose dimension, r, is equal to half the number of off-diagonal bands in the linear system, $(w-1)/2$. For a tridiagonal system ($w = 3$), $\widetilde{\boldsymbol{D}}_i$ is 1×1, and for a penta-diagonal system ($w = 5$), $\widetilde{\boldsymbol{D}}_i$ is 2×2, etc. $\widetilde{\boldsymbol{L}}_i$ and $\widetilde{\boldsymbol{U}}_i$ are short, fat blocks, and $\boldsymbol{L}_i$ and $\boldsymbol{U}_i$ are tall, skinny blocks. $\boldsymbol{D}_i$ is a large, square, non-cyclic, banded block.

According to this grouping strategy, two equations are formed within each partition.

$$\widetilde{\boldsymbol{L}}_i \boldsymbol{x}_{i-1} + \widetilde{\boldsymbol{D}}_i \widetilde{\boldsymbol{x}}_i + \widetilde{\boldsymbol{U}}_i \boldsymbol{x}_i = \widetilde{\boldsymbol{b}}_i \tag{7}$$

$$\boldsymbol{L}_i \widetilde{\boldsymbol{x}}_i + \boldsymbol{D}_i \boldsymbol{x}_i + \boldsymbol{U}_i \widetilde{\boldsymbol{x}}_{i+1} = \boldsymbol{b}_i \tag{8}$$

Assuming $\boldsymbol{D}_i$ is invertible – which is true for the linear systems formed from compact schemes – then $\boldsymbol{x}_i$ can be obtained if both $\widetilde{\boldsymbol{x}}_i$ and $\widetilde{\boldsymbol{x}}_{i-1}$ are known.

$$\boldsymbol{x}_i = \boldsymbol{D}_i^{-1} \left[\boldsymbol{b}_i - \boldsymbol{L}_i \widetilde{\boldsymbol{x}}_i - \boldsymbol{U}_i \widetilde{\boldsymbol{x}}_{i+1} \right] \tag{9}$$

Following the logic of the cyclic reduction, Equation (9) can be used to eliminate $\boldsymbol{x}_{i-1}$ and $\boldsymbol{x}_i$ in Equation (7), which forms the reduced system in Equation (10).

$$\widehat{\boldsymbol{L}}_i \widetilde{\boldsymbol{x}}_{i-1} + \widehat{\boldsymbol{D}}_i \widetilde{\boldsymbol{x}}_i + \widehat{\boldsymbol{U}}_i \widetilde{\boldsymbol{x}}_{i+1} = \widehat{\boldsymbol{b}}_i \tag{10}$$

where

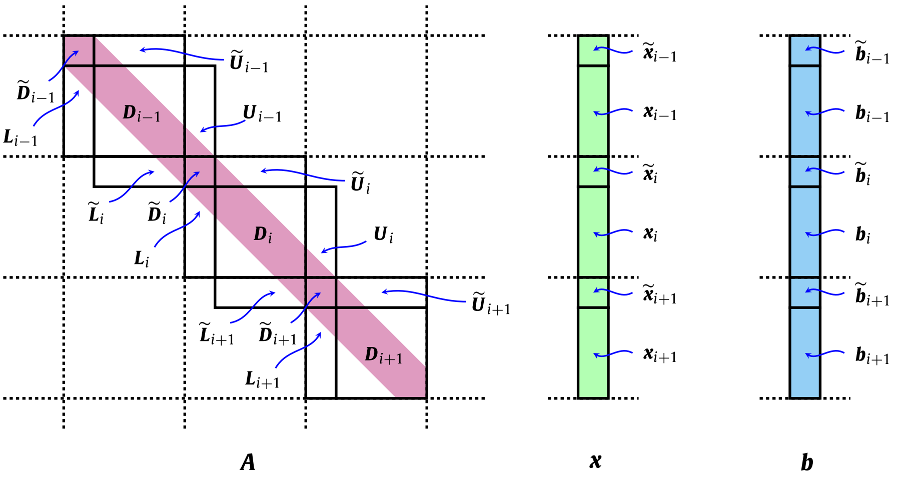

Fig. 4. Partitioned linear system.

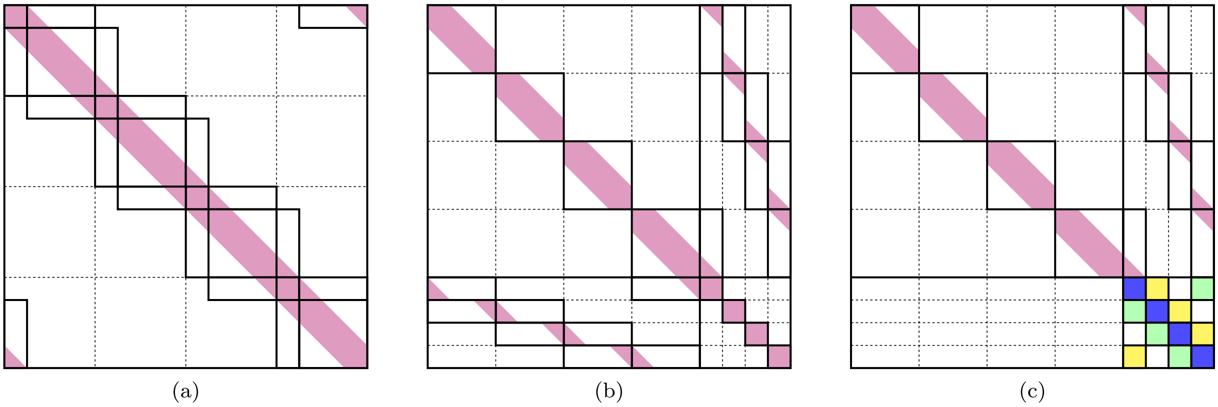

Fig. 5. Sparsity patterns of the system during permutation and block LU-factorization. (a) is the original matrix $\boldsymbol{A}$; (b) is the permuted matrix $\boldsymbol{PAP}^T$; and (c) is the block upper triangular matrix obtained via the block LU-factorization from $\boldsymbol{PAP}^T$.

$$\widehat{\boldsymbol{L}}_i = -\widetilde{\boldsymbol{L}}_i \boldsymbol{D}_{i-1}^{-1} \boldsymbol{L}_{i-1} \tag{11}$$

$$\widehat{\boldsymbol{D}}_i = \widetilde{\boldsymbol{D}}_i - \widetilde{\boldsymbol{L}}_i \boldsymbol{D}_{i-1}^{-1} \boldsymbol{U}_{i-1} - \widetilde{\boldsymbol{U}}_i \boldsymbol{D}_i^{-1} \boldsymbol{L}_i \tag{12}$$

$$\widehat{\boldsymbol{U}}_i = -\widetilde{\boldsymbol{U}}_i \boldsymbol{D}_i^{-1} \boldsymbol{U}_i \tag{13}$$

$$\widehat{\boldsymbol{b}}_i = \widetilde{\boldsymbol{b}}_i - \widetilde{\boldsymbol{L}}_i \boldsymbol{D}_{i-1}^{-1} \boldsymbol{b}_{i-1} - \widetilde{\boldsymbol{U}}_i \boldsymbol{D}_i^{-1} \boldsymbol{b}_i \tag{14}$$

Equation (10) can be represented as $\widehat{\boldsymbol{A}}\widetilde{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$, where $\widehat{\boldsymbol{A}}$ is a block tridiagonal system. If $\boldsymbol{A}$ is cyclic, then $\widehat{\boldsymbol{A}}$ is also cyclic. Considering the grid decomposition strategy, each block in $\widetilde{\boldsymbol{x}}_i$ or $\widehat{\boldsymbol{b}}_i$ is stored across the distributed memory, and each block can be solved efficiently with PCR. This data storage pattern is favorable for PCR, because the blocks can be easily located by the rank of the aligned grid decomposition to conduct the data transfer across the distributed memory. Once the reduced system is solved, all the $\widetilde{\boldsymbol{x}}_i$ are known, and the results can be propagated backward to solve $\boldsymbol{x}_i$ in parallel.

The method can be also interpreted as a block LU-factorization, analogous to the illustration in Gander and Golub [19]. Introducing a permutation matrix $\boldsymbol{P}$, the linear system, $\boldsymbol{Ax} = \boldsymbol{b}$, can be modified to $(\boldsymbol{PAP}^T)(\boldsymbol{Px}) = \boldsymbol{Pb}$, where the row and column permutations, $\boldsymbol{PAP}^T$, regroup $\boldsymbol{D}_i$ and $\widetilde{\boldsymbol{D}}_i$ respectively. The resulting pattern is shown in Fig. 5b. The $\boldsymbol{D}_i$ blocks remain in the top left region on the diagonal, and the $\widetilde{\boldsymbol{D}}_i$ blocks are moved to the bottom right region also on the diagonal. Correspondingly, the $\widetilde{\boldsymbol{L}}_i$ and $\widetilde{\boldsymbol{U}}_i$ blocks show up in the bottom left region, and $\boldsymbol{L}_i$ and $\boldsymbol{U}_i$ blocks are placed in the top right region. The process to obtain Equation (10) is block Gaussian elimination. As a result, the permuted system becomes a block upper triangular system as shown in Fig. 5c, and the reduced system $\widehat{\boldsymbol{A}}$ is formed as the last diagonal block. Additionally, it is clearly shown in Fig. 5c that the top left region only contains the diagonally located blocks, $\boldsymbol{D}_i$. All the non-diagonal blocks are coupled $\boldsymbol{D}_i$ with $\widehat{\boldsymbol{A}}$ only, and no coupling is created among different $\boldsymbol{D}_i$ blocks. This reaffirms that once the reduced system, $\widehat{\boldsymbol{A}}\widetilde{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$, is solved, then the remaining sub-systems, formed by Equation (8), can be solved in parallel on each data partition.

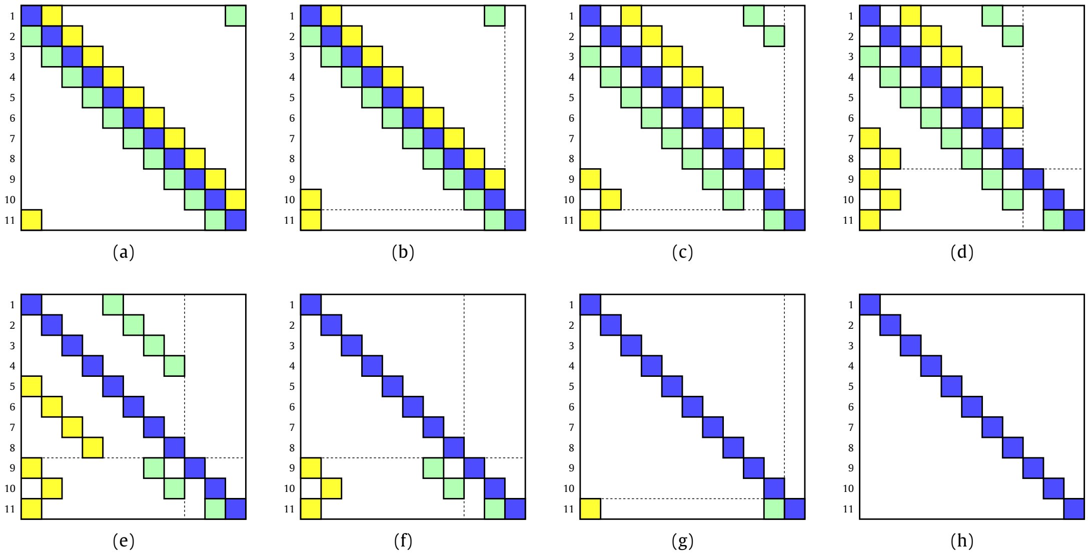

Fig. 6. Reduction procedure of an 11 × 11 $\widehat{\boldsymbol{A}}$. From (a) to (b), row 11 is detached from the sub-system; from (b) to (c), two sub-systems are formed by a PCR step; from (c) to (d), row 9 and row 10 are detached from the sub-systems; from (d) to (f), all the eight unknowns in the sub-systems are solved; from (f) to (g), solutions backward propagate to the first level to solve row 9 and row 10; from (g) to (h), solutions backwards propagate to the root level to solve row 11.

The following section discusses the solution method of the reduced system, $\widehat{\boldsymbol{A}}\widetilde{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$. As aforementioned, $\widehat{\boldsymbol{A}}$ is a block tridiagonal system, which may be cyclic depending on the original banded system, $\boldsymbol{A}$. The block size depends on the half band width of $\boldsymbol{A}$, and the dimension of $\widehat{\boldsymbol{A}}$ equals the number of the aligned grid partitions. The "dimension" of $\widehat{\boldsymbol{A}}$ refers to the number of blocks in each row and column in $\widehat{\boldsymbol{A}}$. Each block in $\widetilde{\boldsymbol{x}}$ and $\widehat{\boldsymbol{b}}$ are stored in a unique partition. With non-periodic boundaries, $\widehat{\boldsymbol{A}}$ is acyclic, and the solution method will follow the block PCR in a fairly straightforward way. With periodic boundaries, $\widehat{\boldsymbol{A}}$ is cyclic, so a non-zero block will show up in the top right and bottom left corners. In this case, if the dimension of $\widehat{\boldsymbol{A}}$ is a power of two, PCR can be directly applied. PCR can still be applied for cyclic $\widehat{\boldsymbol{A}}$ of arbitrary dimension using special treatment. Sweet, in his work [24], suggests such a treatment for cyclic block tridiagonal systems. However, considering the complexity of data storage and data migration, a different treatment is proposed in this paper which requires the dimension of a sub-system of $\widehat{\boldsymbol{A}}$ undergoing a PCR step to be even. If the dimension is odd, a detaching step is needed before the PCR step. During the detaching step, the last row of each sub-system will be used to eliminate the upper and lower off-diagonal blocks of the previous row and the first row of the same sub-system respectively, and then detached from the sub-system. For periodicity, the lower diagonal block in the first row is placed in the last column. After this step, the dimension of each sub-system is an even number, which is ready for the next PCR step. The detached rows will then be addressed and reattached to the sub-system through a backward substitution phase after the rows are solved.

An example is provided by setting $\widehat{\boldsymbol{A}}$ to be an 11 × 11 cyclic tridiagonal matrix. The sparsity pattern in each step is visualized in Fig. 6, and the communication pattern is shown in Fig. 7. On the root level, the number of sub-systems is 1, and the dimension is 11. Since the dimension of this sub-system is odd, the last row needs to detach from the sub-system before conducting PCR. Use the last row to eliminate the upper off-diagonal element of the tenth row and the lower off-diagonal element of the first row, so that a 10 × 10 sub-system is created and the last row is detached, as shown in Fig. 6b. After a PCR step, the 10 × 10 sub-system is split into two 5 × 5 sub-systems on the first level, as shown in Fig. 6c. Before conducting PCR on the first level, the last row of each of the two sub-systems (row 9 and row 10) needs to be detached. Row 9 is used to eliminate the upper off-diagonal element of row 7 (the second to last row of its sub-system on this level) and the lower off-diagonal element of row 1. Row 10 is used to eliminate the upper off-diagonal element of row 8 and the lower diagonal element of row 2 (the first row of its sub-system on this level), so two sub-systems are reduced to 4 × 4 as shown in Fig. 6d. Starting from this level, the number of rows involved in the remaining PCR steps is eight, which is a power of two. At this point, no further detachment is needed, and all the eight unknowns can be solved by two steps of PCR. Then, the eight solutions are backwards substituted into the two 5 × 5 sub-systems on the first level to solve row 9 and row 10. In the final step, the ten solutions propagate backwards to the root level, and are substituted into the 11 × 11 system to solve row 11, so that all the unknowns are solved.

4. Implementation details

The terms $\boldsymbol{D}_i^{-1}\boldsymbol{L}_i$, $\boldsymbol{D}_i^{-1}\boldsymbol{U}_i$, and $\boldsymbol{D}_i^{-1}\boldsymbol{b}_i$, in Equations (11)–(14), are computed by solving the following linear systems.

$$\boldsymbol{D}_i\boldsymbol{S}_i = \boldsymbol{L}_i \tag{15}$$

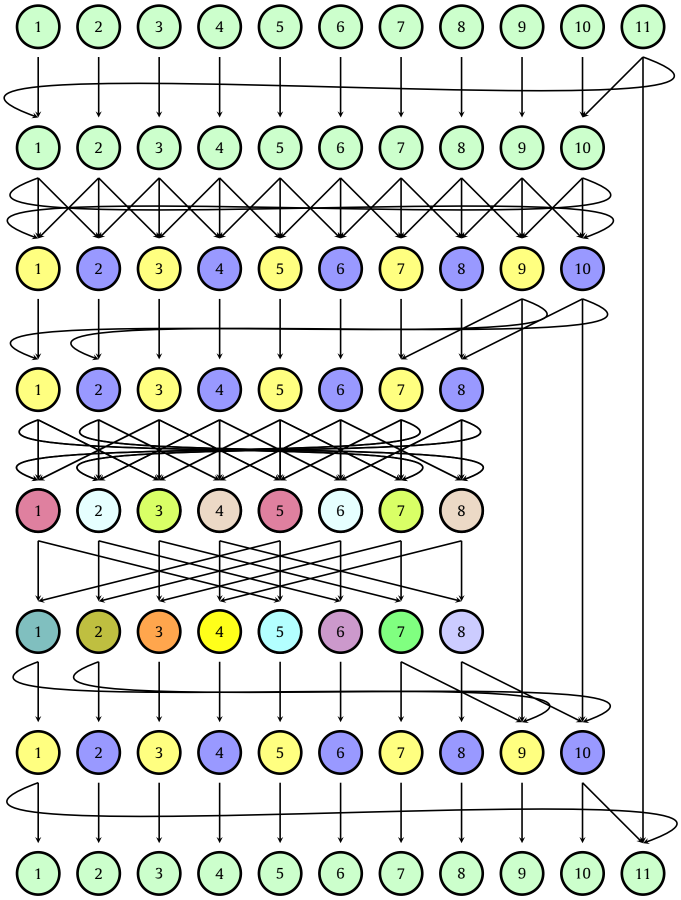

Fig. 7. Communication pattern of PCR for an 11 × 11 cyclic tridiagonal system. The sub-systems in each step are grouped by the same colors. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$\boldsymbol{D}_i \boldsymbol{R}_i = \boldsymbol{U}_i \tag{16}$$

$$\boldsymbol{D}_i \boldsymbol{y}_i = \boldsymbol{b}_i \tag{17}$$

for $\boldsymbol{S}_i$, $\boldsymbol{R}_i$, and $\boldsymbol{y}_i$, respectively. Based on the proposed approach, $\boldsymbol{D}_i$ is an acyclic, compact banded matrix, and all the data on the right-hand side and the unknowns to be solved are stored in the same partition. Therefore, generalized PCR can be used to further parallelize these solves. Using generalized PCR to solve $\boldsymbol{S}_i$, $\boldsymbol{R}_i$, and $\boldsymbol{y}_i$, the number of the parallel reduction steps for each system is $\lceil \log_2 N_i \rceil$, where N_i is the dimension of $\boldsymbol{D}_i$. All the operations at this stage are conducted on the shared memory simultaneously on each partition. Substituting $\boldsymbol{S}_i$, $\boldsymbol{R}_i$, and $\boldsymbol{y}_i$ into Equations (11)–(14), the reduced system – Equation (10) – can be practically constructed according to the following equations.

$$\widehat{\boldsymbol{L}}_i = -\widetilde{\boldsymbol{L}}_i \boldsymbol{S}_{i-1} \tag{18}$$

$$\widehat{\boldsymbol{D}}_i = \widetilde{\boldsymbol{D}}_i - \widetilde{\boldsymbol{L}}_i \boldsymbol{R}_{i-1} - \widetilde{\boldsymbol{U}}_i \boldsymbol{S}_i \tag{19}$$

$$\widehat{\boldsymbol{U}}_i = -\widetilde{\boldsymbol{U}}_i \boldsymbol{R}_i \tag{20}$$

$$\widehat{\boldsymbol{b}}_i = \widetilde{\boldsymbol{b}}_i - \widetilde{\boldsymbol{L}}_i \boldsymbol{y}_{i-1} - \widetilde{\boldsymbol{U}}_i \boldsymbol{y}_i \tag{21}$$

Following the proposed approach to solve for $\widetilde{\boldsymbol{x}}_i$ and substituting into Equation (9), $\boldsymbol{x}_i$ can be obtained by the following operation.

$$\boldsymbol{x}_i = \boldsymbol{y}_i - \boldsymbol{S}_i \widetilde{\boldsymbol{x}}_i - \boldsymbol{R}_i \widetilde{\boldsymbol{x}}_{i+1} \tag{22}$$

A sample implementation is shown in Algorithm 1, where the detaching step, block PCR step, and reattaching step is shown in Algorithm 2, 3, and 4, respectively. The sample code is given in the MPI (message passing interface) style where the rank of partition starts from zero.

Throughout the solution process, the terms $\widetilde{\boldsymbol{L}}_i \boldsymbol{S}_{i-1}$, $\widetilde{\boldsymbol{L}}_i \boldsymbol{R}_{i-1}$, and $\widetilde{\boldsymbol{L}}_i \boldsymbol{y}_{i-1}$ require data transfer from partition $i-1$ to i, and the term $\boldsymbol{R}_i \widetilde{\boldsymbol{x}}_{i+1}$ implies the data transfer from partition $i+1$ to i. It is important to emphasize that the sparsity pattern

Algorithm 1: Implementation of in-place solver following the partitioning shown in Fig. 4. The index i is the rank of partition of the distributed memory sub-group involved in the linear system and is zero-based. The number of distributed memory chunks in the linear system is given by p.

```
in     : $\boldsymbol{D}_i$, $\widetilde{\boldsymbol{L}}_i$, $\widetilde{\boldsymbol{U}}_i$, $i$, $p$
in/out: $\boldsymbol{W}_i \leftarrow \widetilde{\boldsymbol{D}}_i$, $\boldsymbol{Y}_i \leftarrow \boldsymbol{L}_i$, $\boldsymbol{Z}_i \leftarrow \boldsymbol{U}_i$, $\boldsymbol{x}_i \leftarrow \boldsymbol{b}_i$, $\widetilde{\boldsymbol{x}}_i \leftarrow \widetilde{\boldsymbol{b}}_i$

/* Factorization                                                    */
$\boldsymbol{Y}_i \leftarrow$ generalizedPCR($\boldsymbol{D}_i, \boldsymbol{Y}_i$);
$\boldsymbol{Z}_i \leftarrow$ generalizedPCR($\boldsymbol{D}_i, \boldsymbol{Z}_i$);
tag $\leftarrow$ sendToPartition(send_buffer $= \{\boldsymbol{Y}_i, \boldsymbol{Z}_i\}$, dest_rank $= (i+1) \mod p$);
$\widehat{\boldsymbol{U}}_i \leftarrow -\widetilde{\boldsymbol{U}}_i \boldsymbol{Z}_i$;
$\{\boldsymbol{Y}_{i-1}, \boldsymbol{Z}_{i-1}\} \leftarrow$ getFromPartition(tag, src_rank $= (p+i-1) \mod p$);
$\widehat{\boldsymbol{L}}_i \leftarrow -\widetilde{\boldsymbol{L}}_i \boldsymbol{Y}_{i-1}$;
$\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \widetilde{\boldsymbol{L}}_i \boldsymbol{Z}_{i-1} - \widetilde{\boldsymbol{U}}_i \boldsymbol{Y}_i$;

/* Solve reduced system                                             */
$\boldsymbol{x}_i \leftarrow$ generalizedPCR($\boldsymbol{D}_i, \boldsymbol{x}_i$);
tag $\leftarrow$ sendToPartition(send_buffer $= \boldsymbol{x}_i$, dest_rank $= (i+1) \mod p$);
$\boldsymbol{x}_{i-1} \leftarrow$ getFromPartition(tag, src_rank $= (p+i-1) \mod p$);
$\widetilde{\boldsymbol{x}}_i \leftarrow \widetilde{\boldsymbol{x}}_i - \widetilde{\boldsymbol{L}}_i \boldsymbol{x}_{i-1} - \widetilde{\boldsymbol{U}}_i \boldsymbol{x}_i$;

$s \leftarrow 1$; // stride as well as the number of sub-systems
$n_0 \leftarrow p$; // size of each sub-system in the current PCR step
$n_a \leftarrow p$; // number of attached rows in the PCR step $n_a \equiv s \times n_0$
$\mathcal{S} \leftarrow$ initEmptyStack(); // a stack of boolean
while $n_0 > 1$ do
    $\mathcal{S} \leftarrow$ stackPush($n_0 \mod 2 > 0$);
    if $n_0 \mod 2 > 0$ then
        $n_0 \leftarrow n_0 - 1$;
        $n_a \leftarrow n_a - s$;
        Detach the last row of each sub-system (See Algorithm 2);
    end
    Block PCR step (See Algorithm 3);
    $s \leftarrow s \times 2$;
    $n_0 \leftarrow n_0/2$;
end
if $i < n_a$ then
    $\widetilde{\boldsymbol{x}}_i \leftarrow \boldsymbol{W}_i^{-1} \widetilde{\boldsymbol{x}}_i$;
end
while isNotEmpty($\mathcal{S}$) do
    $n_0 \leftarrow n_0 \times 2$;
    $s \leftarrow s/2$;
    if stackPop($\mathcal{S}$) then
        Reattach the last row of each sub-system (See Algorithm 4);
        $n_a \leftarrow n_a + s$;
        $n_0 \leftarrow n_0 + 1$;
    end
end
tag $\leftarrow$ sendToPartition(send_buffer $= \widetilde{\boldsymbol{x}}_i$, dest_rank $= (p+i-1) \mod p$);
$\widetilde{\boldsymbol{x}}_{i+1} \leftarrow$ getFromPartition(tag, src_rank $= (i+1) \mod p$);
$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \boldsymbol{Y}_i \widetilde{\boldsymbol{x}}_i - \boldsymbol{Z}_i \widetilde{\boldsymbol{x}}_{i+1}$;
```

of the matrix $\widetilde{\boldsymbol{L}}_i$ results in only a fraction of the allocated data in $\boldsymbol{y}_{i-1}$, $\boldsymbol{R}_{i-1}$, and $\boldsymbol{S}_{i-1}$ exchanged across neighboring data partitions as illustrated in Fig. 5b and Fig. 8a. For the banded matrix, $\boldsymbol{A}$, with a bandwidth $w = 2r+1$, only the last r columns in $\widetilde{\boldsymbol{L}}_i$ are non-trivial. Therefore, only the last r rows in $\boldsymbol{S}_{i-1}$, $\boldsymbol{R}_{i-1}$, and $\boldsymbol{y}_{i-1}$ are needed for neighboring communication. Similarly, the matrix products involving $\widetilde{\boldsymbol{U}}_i$ can be computed very efficiently due to its sparsity pattern as shown in Fig. 8b. If the number of rows in each partition is much larger than the system bandwidth ($N_i \gg r$), a significant reduction of data size for communication and multiplication can be achieved. The reduced system $\widehat{\boldsymbol{A}}\widetilde{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$ (Equation (10)) is solved on distributed memory, and each parallel reduction step requires data communication between neighboring partitions. If $\boldsymbol{A}$ is acyclic, then $\widehat{\boldsymbol{A}}$ can be solved with the classic block PCR, although the proposed algorithm can still be used by setting the cyclic entries to zero, and the number of the parallel reduction steps is $\lceil \log_2 p \rceil$, where p is the number of partitions. If $\boldsymbol{A}$ is cyclic, using the proposed algorithm, the number of the parallel reduction steps is $\lfloor \log_2 p \rfloor$. In addition, if p is not a power of 2, the number of rows that are involved in the detaching and reattaching throughout the solving process equals $p - 2^{\lfloor \log_2 p \rfloor}$, and the numbers of the parallel detaching and reattaching steps are $\left\{ \sum_{n=0}^{\lfloor \log_2 p \rfloor} \left(\lfloor 2^{-n} p \rfloor \mod 2 \right) \right\} - 1$.

In the motivating applications, such as evaluating derivatives using compact finite differences in a multiphysics application, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is frequently solved with varying $\boldsymbol{b}$ but constant $\boldsymbol{A}$. Noticing that the construction of $\widehat{\boldsymbol{L}}_i$, $\widehat{\boldsymbol{D}}_i$, $\widehat{\boldsymbol{U}}_i$, and $\boldsymbol{D}_i$, does not require the right-hand side, $\boldsymbol{b}$, such construction is needed only once, and the original matrix can be pre-factorized. During the pre-factorization, the reduction coefficients on each stage, k_{+j} and k_{-j}, and the information needed to solve

Algorithm 2: Detaching process in Algorithm 1.

```
if $n_a \le i < (n_a + s)$ then
    tag_a ← sendToPartition(send_buffer = $\{\widehat{\boldsymbol{L}}_i, \boldsymbol{W}_i, \widehat{\boldsymbol{U}}_i, \widetilde{\boldsymbol{x}}_i\}$, dest_rank = $i - s$);
    tag_b ← sendToPartition(send_buffer = $\{\widehat{\boldsymbol{L}}_i, \boldsymbol{W}_i, \widehat{\boldsymbol{U}}_i, \widetilde{\boldsymbol{x}}_i\}$, dest_rank = $i - n_a$);
end
if $n_a \le (i + s) < (n_a + s)$ then
    $\{\widehat{\boldsymbol{L}}_{i+s}, \boldsymbol{W}_{i+s}, \widehat{\boldsymbol{U}}_{i+s}, \widetilde{\boldsymbol{x}}_{i+s}\}$ ← getFromPartition(tag_a, src_rank = $i + s$);
    $\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \widehat{\boldsymbol{U}}_i \boldsymbol{W}_{i+s}^{-1} \widehat{\boldsymbol{L}}_{i+s}$;
    $\widetilde{\boldsymbol{x}}_i \leftarrow \widetilde{\boldsymbol{x}}_i - \widehat{\boldsymbol{U}}_i \boldsymbol{W}_{i+s}^{-1} \widetilde{\boldsymbol{x}}_{i+s}$;
    $\widehat{\boldsymbol{U}}_i \leftarrow -\widehat{\boldsymbol{U}}_i \boldsymbol{W}_{i+s}^{-1} \widehat{\boldsymbol{U}}_{i+s}$;
end
if $n_a \le (i + n_a) < (n_a + s)$ then
    $\{\widehat{\boldsymbol{L}}_{i-s}, \boldsymbol{W}_{i-s}, \widehat{\boldsymbol{U}}_{i-s}, \widetilde{\boldsymbol{x}}_{i-s}\}$ ← getFromPartition(tag_b, src_rank = $i + n_a$);
    $\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \widehat{\boldsymbol{L}}_i \boldsymbol{W}_{i-s}^{-1} \widehat{\boldsymbol{U}}_{i-s}$;
    $\widetilde{\boldsymbol{x}}_i \leftarrow \widetilde{\boldsymbol{x}}_i - \widehat{\boldsymbol{L}}_i \boldsymbol{W}_{i-s}^{-1} \widetilde{\boldsymbol{x}}_{i-s}$;
    $\widehat{\boldsymbol{L}}_i \leftarrow -\widehat{\boldsymbol{L}}_i \boldsymbol{W}_{i-s}^{-1} \widehat{\boldsymbol{L}}_{i-s}$;
end
```

Algorithm 3: Block PCR process in Algorithm 1.

```
if $i < n_a$ then
    tag_a ← sendToPartition(send_buffer = $\{\widehat{\boldsymbol{L}}_i, \boldsymbol{W}_i, \widehat{\boldsymbol{U}}_i, \widetilde{\boldsymbol{x}}_i\}$, dest_rank = $(n_a + i - s) \mod n_a$);
    tag_b ← sendToPartition(send_buffer = $\{\widehat{\boldsymbol{L}}_i, \boldsymbol{W}_i, \widehat{\boldsymbol{U}}_i, \widetilde{\boldsymbol{x}}_i\}$, dest_rank = $(i + s) \mod n_a$);
    $\{\widehat{\boldsymbol{L}}_{i+s}, \boldsymbol{W}_{i+s}, \widehat{\boldsymbol{U}}_{i+s}, \widetilde{\boldsymbol{x}}_{i+s}\}$ ← getFromPartition(tag_a, src_rank = $(i + s) \mod n_a$);
    $\{\widehat{\boldsymbol{L}}_{i-s}, \boldsymbol{W}_{i-s}, \widehat{\boldsymbol{U}}_{i-s}, \widetilde{\boldsymbol{x}}_{i-s}\}$ ← getFromPartition(tag_b, src_rank = $(n_a + i - s) \mod n_a$);
    $\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \widehat{\boldsymbol{U}}_i \boldsymbol{W}_{i+s}^{-1} \widehat{\boldsymbol{L}}_{i+s} - \widehat{\boldsymbol{L}}_i \boldsymbol{W}_{i-s}^{-1} \widehat{\boldsymbol{U}}_{i-s}$;
    $\widetilde{\boldsymbol{x}}_i \leftarrow \widetilde{\boldsymbol{x}}_i - \widehat{\boldsymbol{U}}_i \boldsymbol{W}_{i+s}^{-1} \widetilde{\boldsymbol{x}}_{x+s} - \widehat{\boldsymbol{L}}_i \boldsymbol{W}_{i-s}^{-1} \widetilde{\boldsymbol{x}}_{i-s}$;
    $\widehat{\boldsymbol{L}}_i \leftarrow -\widehat{\boldsymbol{L}}_i \boldsymbol{W}_{i-s}^{-1} \widehat{\boldsymbol{L}}_{i-s}$;
    $\widehat{\boldsymbol{U}}_i \leftarrow -\widehat{\boldsymbol{U}}_i \boldsymbol{W}_{i+s}^{-1} \widehat{\boldsymbol{U}}_{i+s}$;
end
```

Algorithm 4: Reattaching process in Algorithm 1.

```
if $n_a \le (i + s) < (n_a + s)$ then
    tag_b ← sendToPartition(send_buffer = $\widetilde{\boldsymbol{x}}_i$, dest_rank = $i + s$);
end
if $n_a \le (i + n_a) < (n_a + s)$ then
    tag_a ← sendToPartition(send_buffer = $\widetilde{\boldsymbol{x}}_i$, dest_rank = $i + n_a$);
end
if $n_a \le i < (n_a + s)$ then
    $\widetilde{\boldsymbol{x}}_{i+s}$ ← getFromPartition(tag_a, src_rank = $i - n_a$);
    $\widetilde{\boldsymbol{x}}_{i-s}$ ← getFromPartition(tag_b, src_rank = $i - s$);
    $\widetilde{\boldsymbol{x}}_i \leftarrow \widetilde{\boldsymbol{x}}_i - \widehat{\boldsymbol{L}}_i \widetilde{\boldsymbol{x}}_{i-s} - \widehat{\boldsymbol{U}}_i \widetilde{\boldsymbol{x}}_{i+s}$;
    $\widetilde{\boldsymbol{x}}_i \leftarrow \boldsymbol{W}_i^{-1} \widetilde{\boldsymbol{x}}_i$;
end
```

$\widehat{\boldsymbol{A}}\widetilde{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$ can be calculated and stored. During the solution process, Equation (17) and Equation (21) are needed to construct the right-hand side of the reduced system to solve $\widetilde{\boldsymbol{x}}_i$. Finally, Equation (22) is used to solve for $\boldsymbol{x}_i$.

5. Performance

5.1. Asymptotic performance analysis

This section describes the asymptotic analysis of compute and communication scaling for the proposed algorithm. As described in Section 3 and 4, consider the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{A}$ is an $N \times N$ cyclic compact banded matrix with bandwidth $2r + 1$. $\boldsymbol{x}$ and $\boldsymbol{b}$ are $N \times M$ matrices representing M independent solutions and right-hand-side vectors respectively. This system is partitioned across p distributed memory processes and the local system for process i, $\boldsymbol{D}_i$, is of size $N_i \times N_i$.

The algorithm can be grouped into four phases. Phases I and II are the local and coupled forward elimination phases respectively. Phase III is the distributed solve of the reduced system, and phase IV is the concurrent backward substitution phase. Table 1 shows the asymptotic scaling of the dominant compute and communication costs per process for each phase of the algorithm. The table shows that the explicit p dependence is only in phase III which computes the solution of the reduced system. However, depending on the scaling regime and partitioning strategy, other hidden p dependence might also be important.

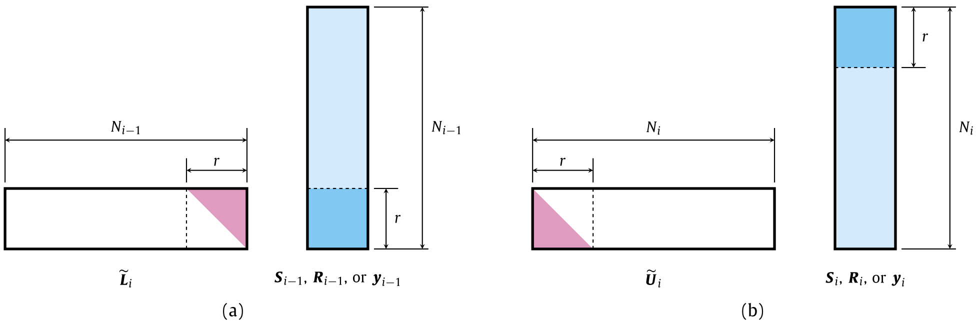
N_{i-1}
r
N_{i-1}
r
$\widetilde{\boldsymbol{L}}_i$
$\boldsymbol{S}_{i-1}$, $\boldsymbol{R}_{i-1}$, or $\boldsymbol{y}_{i-1}$
(a)
N_i
r
r
N_i
$\widetilde{\boldsymbol{U}}_i$
$\boldsymbol{S}_i$, $\boldsymbol{R}_i$, or $\boldsymbol{y}_i$
(b)

Fig. 8. Communication and multiplication patterns: (a) $\widetilde{\boldsymbol{L}}_i\boldsymbol{S}_{i-1}$, $\widetilde{\boldsymbol{L}}_i\boldsymbol{R}_{i-1}$, and $\widetilde{\boldsymbol{L}}_i\boldsymbol{y}_{i-1}$; (b) $\widetilde{\boldsymbol{U}}_i\boldsymbol{S}_i$, $\widetilde{\boldsymbol{U}}_i\boldsymbol{R}_i$, and $\widetilde{\boldsymbol{U}}_i\boldsymbol{y}_i$.

Table 1
Asymptotic scaling of computation and communication of each phase in the solution process.

Phase	Equations	Computation cost	Communication cost
I	(15) (16) (17)	$\mathcal{O}\left[N_i \log_2 N_i \max(r, M)\right]$	0
II	(18) (19) (20) (21)	$\mathcal{O}\left[r^2 \max(r, M)\right]$	$\mathcal{O}\left[r \max(r, M)\right]$
III	(10)	$\mathcal{O}\left[r^2 M \log_2 p\right]$	$\mathcal{O}\left[rM \log_2 p\right]$
IV	(22)	$\mathcal{O}\left[N_i r M\right]$	$\mathcal{O}\left[rM\right]$

5.1.1. 1D decomposition

Consider a problem with an $N \times N_0 \times N_0$ grid requiring the solution along the first dimension and partitioned using p processes. The matrix bandwidth can also be considered to be much smaller than the batch size, $r \ll M$, and the size of the local system, $r \ll N_i$. Then, $\max(r, M) = M$. The total cost of computation and communication during the solve process in the asymptotic limit is

$$\text{Computation cost} \sim \mathcal{O}\left(MN_i \log_2 N_i + Mr\left[C_1 N_i + C_2 r \log_2 p\right]\right) \tag{23}$$

$$\text{Communication cost} \sim \mathcal{O}\left(rM\left[C_3 + \log_2 p\right]\right) \tag{24}$$

where C_1, C_2 and C_3 are all $\mathcal{O}(1)$ constants.

For strong scaling, N and $M = N_0^2$ are held constant. From Table 1, the total cost of the algorithm in the strong scaling regime is

$$\text{Computation cost} \sim \mathcal{O}\left(N_0^2 N\left\{\frac{1}{p}\left[C_1 r + \log_2\left(\frac{N}{p}\right)\right] + C_2 \frac{r^2}{N} \log_2 p\right\}\right) \tag{25}$$

$$\text{Communication cost} \sim \mathcal{O}\left(rN_0^2\left[C_3 + \log_2 p\right]\right) \tag{26}$$

For weak scaling, $N_i \approx N/p$ and $M = N_0^2$ remain constant. The total cost of the algorithm in the weak scaling regime is then given by

$$\text{Computation cost} \sim \mathcal{O}\left(N_0^2 N_i\left[C_1 r + \log_2 N_i\right] + C_2 N_0^2 r^2 \log_2 p\right) \tag{27}$$

$$\text{Communication cost} \sim \mathcal{O}\left(rN_0^2\left[C_3 + \log_2 p\right]\right) \tag{28}$$

5.1.2. 3D decomposition

Consider a three dimensional problem with an $N \times N \times N$ grid and partitioned using p processes with $p^{1/3}$ processes per dimension. Then, $N_i \approx N/p^{1/3}$ and $M \approx N_i^2 \approx N^2/p^{2/3}$ are held constant. The matrix bandwidth can also be considered to be much smaller than the batch size, $r \ll M$. As a direct result, $\max(r, M) = M$. From Table 1, the total cost of the algorithm in the weak scaling regime is

$$\text{Computation cost} \sim \mathcal{O}\left(N_i^3\left[C_1 r + \log_2 N_i\right] + \frac{1}{3} C_2 N_i^2 r^2 \log_2 p\right) \tag{29}$$

$$\text{Communication cost} \sim \mathcal{O}\left(rN_i^2\left[C_3 + \frac{1}{3}\log_2 p\right]\right) \tag{30}$$

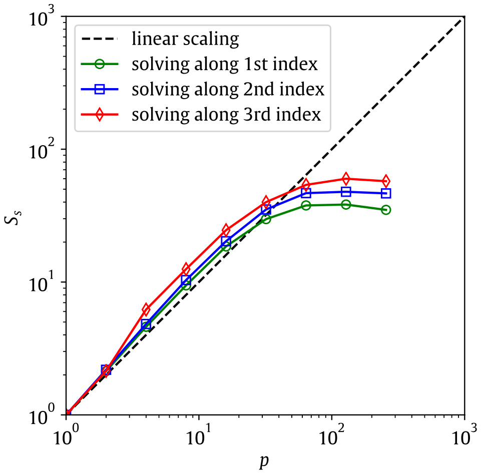

Fig. 9. Measured strong scaling of the linear solver for each of the coordinate indices. The curve for each index is normalized by its own single-GPU time, so all speedups start at unity.

5.2. Measured scaling results

In this section, the performance of the linear solver is demonstrated both in isolation and in the context of a representative fluid mechanics application problem. All tests in this section were performed on the *Summit* supercomputer at the Oak Ridge Leadership Computing Facility (OLCF) at Oak Ridge National Laboratory (ORNL) [31]. Each *Summit* node consists of 6 NVIDIA Tesla V100 GPUs and 2 IBM Power 9 processors. The nodes on the system are connected with Mellanox EDR 100G Infiniband interconnect, arranged in a non-blocking fat tree topology. In the present implementation, all compute operations are conducted on GPUs, while CPUs are dedicated to control and communication tasks. The code for the results in this section was written in C++ using the Kokkos framework [32,33].

For the linear solver alone, both strong and weak scaling results are presented for solving $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is a cyclic tridiagonal system with bands given by $\boldsymbol{A} = \mathcal{B}\,[\ 1/3,\ \ 1,\ \ 1/3\ ]$. This linear system represents the left-hand side of the sixth order compact first derivative scheme on a periodic domain [2]. For all linear solver scaling tests, the linear system is solved 1000 times, and speedup based on the average time is reported. In the strong scaling test, the dimension of $\boldsymbol{A}$ is 8192×8192, and the linear system is solved 256^2 times in parallel, i.e., the dimensions of $\boldsymbol{b}$ and $\boldsymbol{x}$ are 8192×256^2. In the context of the compact finite difference scheme, this is equivalent to computing a spatial derivative along a column of 3D Cartesian grid partitions, where the grid dimension is 8192 along the solving direction and 256×256 perpendicular to the solving direction. For example, when solving along the first index, the grid is $8192 \times 256 \times 256$ mesh. As the number of GPUs used is increased, the domain is decomposed equally along the solving direction so that each partition has the size of $(8192/p) \times 256 \times 256$. When solving along other directions, the dimensions are permuted correspondingly. Some small differences in performance among the directions are expected because of memory striding. In this implementation, right memory layout is used, where the third index maps to contiguous memory. The strong scaling speedup, S_s, is defined as

$$S_s(p) = \frac{T_1}{T_p} \tag{31}$$

where T_p is the wall time when using p GPUs. The strong scaling results for each index direction are shown in Fig. 9.

The strong scaling plot shows speedup from $p = 1$ to $p = 256$. Beyond $p = 64$, the speedup plateaus. This is because in the strong scaling regime, the leading order compute cost decreases linearly with p while the communication cost increases as $\log_2 p$ as described in Section 5.1.1. The speedup between $p = 2$ and $p = 4$ is super-linear and it is suspected that this is mainly due to GPU cache benefits as the problem size per GPU keeps getting smaller. There is another effect that contributes to this super-linear scaling that stems from Equation (25). As p increases, the computation cost initially decreases as $p^{-1} \log_2 p$ contributing to the super-linear scaling behavior. This is only valid in the compute dominant limit of low p and large N, particularly when $r^2/N \ll 1$. As p increases, the cost of solving the reduced system increases causing a reduction in scaling efficiency in addition to the increased communication cost. At $p = 32$, approximately where the three curves cross the linear scaling line, the data chunk on each GPU is $256 \times 256 \times 256$. This is the chunk size used as the basis of the weak scaling tests, which are discussed next.

The weak scaling performance is shown for solving $\boldsymbol{Ax} = \boldsymbol{b}$, with the same matrix $\boldsymbol{A}$ as in the strong scaling test. The computational domain is partitioned along the solve direction into cubic sub-domains of size $N_0 \times N_0 \times N_0$, so $\boldsymbol{A}$ is $pN_0 \times pN_0$, and it is solved N_0^2 times in parallel. For all the weak scaling tests, $N_0 = 256$ is chosen, so that each GPU operates on a chunk of data that is 256^3. The weak scaling results for the isolated linear system solve are presented in two ways in Fig. 10. Here, the weak scaling "speedup", S_w, is reported:

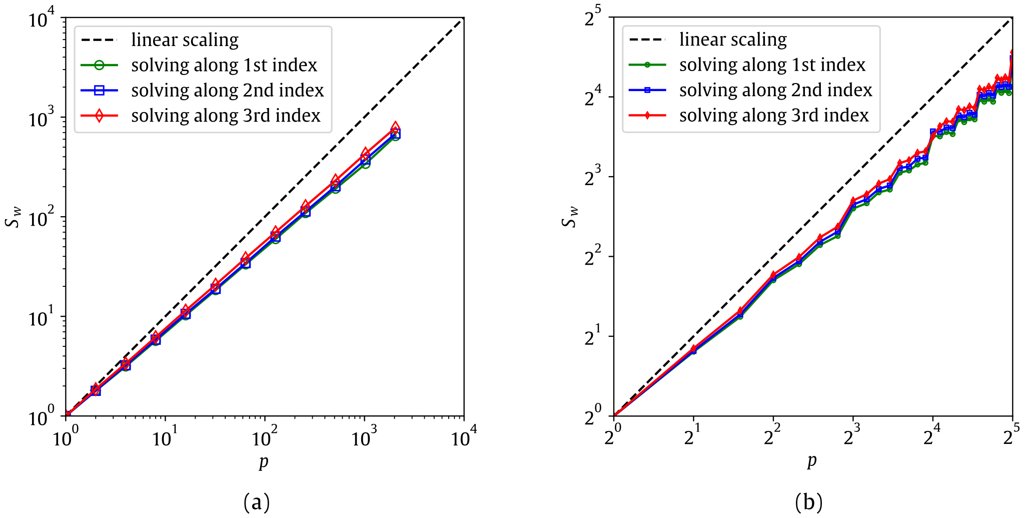

Fig. 10. Measured weak scaling of the linear solver: (a) number of GPUs increasing in powers of 2; (b) number of GPUs increasing linearly. Data is stored on the right memory layout where the 3rd index is the fastest looping index.

$$S_w(p) = \frac{p \times T_1}{T_p} \tag{32}$$

First, in Fig. 10a, the number of GPUs used is always a power of 2, from 1 to 2048. As the number of GPUs is increased in the weak scaling test, the dimension of only one coordinate direction is increased at a time, and all three directions are tested. The grid sizes for the series of tests for the first index, for example, are 256 × 256 × 256, 512 × 256 × 256, 1024 × 256 × 256, etc. Second, in Fig. 10b, the number of GPUs used increases linearly from 1 to 32, to show the effect of a non-ideal problem decomposition on performance. Also, the differences in scaling among the index directions are very small, meaning that in a large scale 3D problem, no one direction will dominate the computational cost. These results show that the scaling of the linear solver is reasonably good up to a very large number of GPUs. For context, the last data point comes from running on 2048 GPUs on *Summit*, or about 8% of the entire machine. This test exercises one coordinate direction at a time on a column of the domain decomposition, comparable to the highlighted partitions in Fig. 1 in order to predict the performance in a realistic computation application. Accordingly, the last data point represents the intended 3D equal size domain decomposition used by the linear solver in a production size simulation which uses 2048^3 GPUs.

These results also show that while the solver scales best when using a number of GPUs equal to a power of 2, its performance is degraded using odd or even prime numbers of GPUs. This occurs because additional work in the form of detach-reattach steps is required when not using a power-of-2 number of GPUs. The worst case scenario in terms of additional work required is to use a number of GPUs equal to $2^n - 1$. This choice requires $n - 1$ steps of PCR and $n - 1$ stages of detach-reattach operations. An additional series of weak scaling results is presented in Fig. 11, which compares the weak scaling performance of using 2^n vs. $2^n - 1$ GPUs. Only the 1st index direction is shown, since the results are qualitatively the same for all directions. Depending on the specific machine and application size, it may not always be practical to use a number of GPUs that is a power of 2. As a result, it is expected that the practical weak scaling behavior of this algorithm lies in the range between the curves in Fig. 11. Both curves are demonstrated to be linear over the range tested. This is expected based on how the number of PCR steps and detach-reattach operations scales with the number of processes. Since the lines have different slopes, this means that the relative benefit of using the ideal number of GPUs becomes greater as the problem size is increased.

Finally, weak scaling is demonstrated on a fluid mechanics application – the direct numerical simulation of a Taylor-Green vortex problem at the Reynolds number of 1600 and Mach number of 0.08 [34] – by using a compressible Navier-Stokes direct numerical simulation solver. The simulations were conducted using the sixth-order staggered compact finite difference schemes and compact interpolators for spatial discretization [16]. The details of the problem description and numerical formulation are illustrated in Appendix A. For each weak scaling test, a constant time step determined by stability requirements was used, and wall time data was collected for 100 time steps. The computational cost is dominated by calculating derivatives and interpolations in the Navier-Stokes equations, which involves solving linear systems similar to the one above. The solution at a representative time is visualized in Fig. A.13. This test is useful because its domain and domain decomposition are much more realistic than those of the isolated linear solver test. Also, it involves approximately equal numbers of linear solves along all three coordinate indices. Finally, it tests whether the linear solver performance enables good scaling on a practical problem. Like the first linear solver test, scaling is reported in powers of 2, but quantities of GPUs of 6×2^n were also tested. This second series corresponds to full utilization of *Summit* nodes, which have 6 GPUs each. The weak scaling results, including both setups, are shown in Fig. 12, which demonstrates excellent scaling up to 24576 GPUs, or 89% of the nodes on *Summit*.

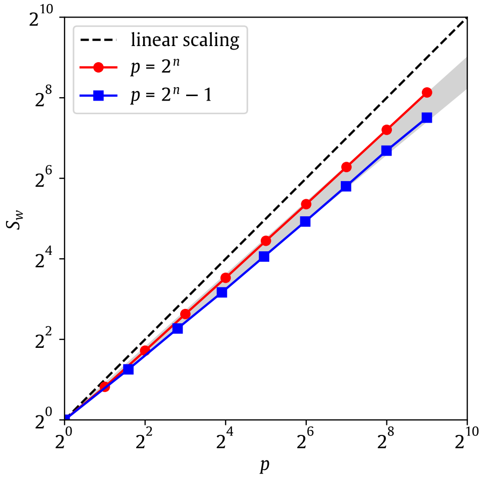

Fig. 11. Measured weak scaling of the linear solver using best-case (2^n) and worst-case ($2^n - 1$) numbers of GPUs, solving along the 1st index.

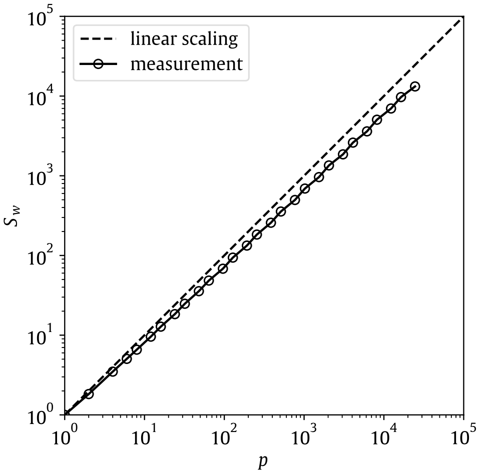

Fig. 12. Measured weak scaling (256^3 grid point per GPU) of a Navier-Stokes solver on the Taylor-Green vortex problem using compact finite difference and interpolation methods. Data is reported using both 4×2^n and 6×2^n GPUs on *Summit*.

The reason that the Taylor-Green vortex problem scales better than the linear solver test is due to the 3D domain decomposition. The linear solver weak scaling tested an extreme scenario, with a 1D domain decomposition. This would only be appropriate for a domain with one dimension much longer than the other two. Such an aspect ratio is not typical for simulations of turbulent flows.

6. Conclusions

In this work, a direct linear solver for compact banded systems is presented and demonstrated to have scalable performance on a petascale GPU platform. The algorithm is applicable for a wide variety of high performance computing platforms with heterogeneous computing capabilities. The sparsity patterns that result in the factorized matrix blocks are leveraged in the overall algorithm to avoid large data transfers across the distributed memory partitions and to reduce the floating point operational cost of matrix-matrix multiplications. As such, the proposed algorithm has significant advantages over conventional strategies that involve "all-to-all" communication patterns. These advantages thereby enable the proposed algorithm to be suitable for distributed heterogeneous computing environments requiring programming paradigms such as "MPI+X", and to reduce the strong performance dependence on the underlying network topology. The weak scalability is shown on a canonical 3D periodic Navier-Stokes problem using compact finite difference and interpolation schemes involving cyclic banded tridiagonal linear systems. The algorithm works on a flexible number of distributed memory partitions and optimal performance is recovered when the number of ranks is a power of two. This work is directly beneficial to the large scale computations of a wide range of partial differential equation problems using compact numerical schemes such as in fluid mechanics, solid mechanics, and electromagnetics.

CRediT authorship contribution statement

Hang Song: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Kristen V. Matsuno:** Formal analysis, Software, Validation, Writing – review & editing. **Jacob R. West:** Formal analysis, Investigation, Software, Writing – review & editing. **Akshay Subramaniam:** Conceptualization, Formal analysis, Methodology, Software, Writing – review & editing. **Aditya S. Ghate:** Software, Validation, Writing – review & editing. **Sanjiva K. Lele:** Project administration, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors are grateful to Professor Eric Darve for helpful discussions. The authors also thank the referees for their constructive comments. This research is supported by the National Science Foundation, grant number NSF-OAC-2103509. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725 [31]. This work also used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562 [35]. This work used XSEDE resources *Bridges* and *Comet* through allocation TG-CCR130001.

Appendix A. Taylor-Green vortex

A.1. Problem description

The Taylor-Green vortex problem is a well-established fluid mechanics problem defined on 3D periodic domain, $\boldsymbol{x} \in [0, 2\pi l) \times [0, 2\pi l) \times [0, 2\pi l)$, where l is a characteristic length. The tests used in this work were conducted by solving the compressible Navier-Stokes equations.

$$\frac{\partial \boldsymbol{\phi}}{\partial t} + \nabla \cdot \boldsymbol{F} + \nabla \cdot \boldsymbol{G} = 0 \tag{A.1}$$

where $\boldsymbol{\phi}$ is the set of the conservative variables; $\boldsymbol{F}$ is the set of inviscid fluxes; and $\boldsymbol{G}$ is the set of diffusive fluxes. They are defined as

$$\boldsymbol{\phi} = \begin{bmatrix} \rho \\ \rho \boldsymbol{u} \\ \rho(e + \boldsymbol{u} \cdot \boldsymbol{u}/2) \end{bmatrix} \tag{A.2}$$

$$\boldsymbol{F} = \begin{bmatrix} \rho \boldsymbol{u} \\ \rho\left(\boldsymbol{u} \otimes \boldsymbol{u}\right) + P\boldsymbol{I} \\ \boldsymbol{u}\left(\rho e + \rho \boldsymbol{u} \cdot \boldsymbol{u}/2 + P\right) \end{bmatrix} \tag{A.3}$$

$$\boldsymbol{G} = \begin{bmatrix} 0 \\ -\boldsymbol{\sigma} \\ \boldsymbol{q} - \boldsymbol{u} \cdot \boldsymbol{\sigma} \end{bmatrix} \tag{A.4}$$

where ρ is the density; $\boldsymbol{u} = [u, v, w]^T$ is the velocity vector; P is the pressure; $\boldsymbol{I}$ is the identity tensor; e is the specific internal energy, $\boldsymbol{\sigma}$ is the viscous stress tensor; and $\boldsymbol{q}$ is the heat flux. The fluid is treated as ideal gas with the following equation of state.

$$P = \rho R T \tag{A.5}$$

where R is the specific gas constant; and T is the temperature. Accordingly, the internal energy is

$$e = \frac{RT}{\gamma - 1} \tag{A.6}$$

where γ is the ratio of specific heats. The viscous stress tensor is modeled as

$$\boldsymbol{\sigma} = \mu\left[(\nabla \boldsymbol{u}) + (\nabla \boldsymbol{u})^T\right] + \left(\beta - \frac{2}{3}\mu\right)(\nabla \cdot \boldsymbol{u})\,\boldsymbol{I} \tag{A.7}$$

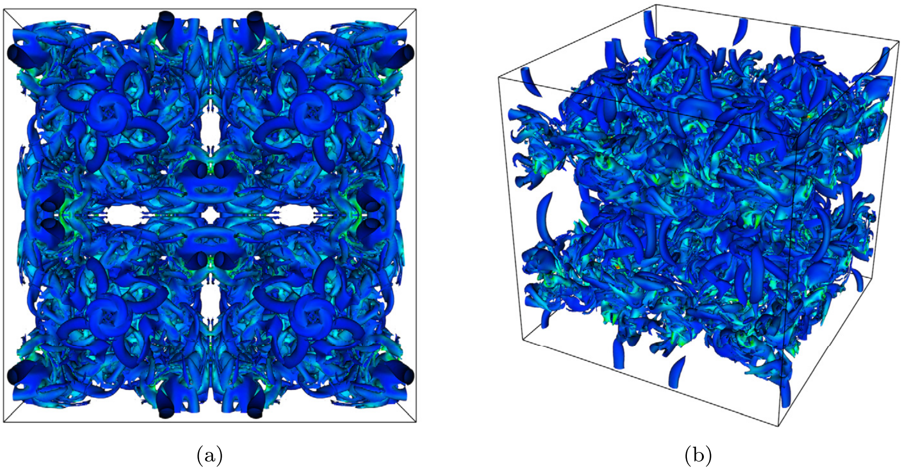

Fig. A.13. Q-criterion iso-surface colored by enstrophy in the Taylor-Green vortex problem using 256^3 points. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

where μ is the dynamic shear viscosity, and β is the bulk viscosity. For the simulations used in this work, $\beta = 0$ and μ is set to be a constant determined from the Reynolds number, Re.

$$\mathrm{Re} = \frac{\rho_0 V l}{\mu} \tag{A.8}$$

where ρ_0 is the mean density as well as the initial density of the fluid, and V is a characteristic velocity. The heat flux $\boldsymbol{q}$ is computed based on the Fourier law

$$\boldsymbol{q} = -\kappa \nabla T \tag{A.9}$$

where κ is the heat conductivity controlled by the Prandtl number, Pr, defined in the following.

$$\mathrm{Pr} = \frac{\gamma R \mu}{(\gamma - 1)\kappa} \tag{A.10}$$

The initial velocity, $[u_0, v_0, w_0]^T$, and pressure, P_0, fields are set as [34]

$$u_0 = V \sin(x/l)\cos(y/l)\cos(z/l) \tag{A.11}$$

$$v_0 = -V \cos(x/l)\sin(y/l)\cos(z/l) \tag{A.12}$$

$$w_0 = 0 \tag{A.13}$$

$$P_0 = P_{\mathrm{ref}} + \frac{\rho_0 V^2}{16}\left[\cos(2x/l) + \cos(2y/l)\right]\left[\cos(2z/l) + 2\right] \tag{A.14}$$

where $l = 1$, $V = 1$, $\rho_0 = 1$, and $P_{\mathrm{ref}} = 100$. The Reynolds number and Prandtl number are set to Re = 1600 and Pr = 0.7, respectively. The specific gas constant is set to unity, and the specific heat ratio $\gamma = 5/3$, so that the Mach number, Ma, consistent with the initial condition, is approximately 0.08, which is calculated in the following based on its definition.

$$\mathrm{Ma} = \frac{V}{\sqrt{\gamma P_{\mathrm{ref}}/\rho_0}} \tag{A.15}$$

A.2. Numerical schemes

The problem is numerically computed on a 3D Cartesian uniform mesh using the staggered sixth order compact finite difference schemes and the sixth order compact interpolators [2,16], as shown in the following two equations.

$$\frac{9}{62} f'_{i-1} + f'_i + \frac{9}{62} f'_{i+1} = \frac{63}{62}\left(\frac{f_{i+1/2} - f_{i-1/2}}{\Delta}\right) + \frac{17}{62}\left(\frac{f_{i+3/2} - f_{i-3/2}}{3\Delta}\right) \tag{A.16}$$

$$\frac{3}{10} f^I_{i-1} + f^I_i + \frac{3}{10} f^I_{i+1} = \frac{3}{2}\left(\frac{f_{i+1/2} + f_{i-1/2}}{2}\right) + \frac{1}{10}\left(\frac{f_{i+3/2} + f_{i-3/2}}{2}\right) \tag{A.17}$$

where f, f', and f^I represent the original field, first derivative, and interpolated field, respectively; the subscript indicates the grid index in the corresponding direction; and Δ is the grid space in the corresponding direction. The primitive variables are all stored at the grid collocations, and all the fluxes in $\boldsymbol{F}$ and $\boldsymbol{G}$ are constructed at the staggered locations in the corresponding directions. The time advancement uses the standard fourth order Runge-Kutta method.

References

[1] T. Colonius, S.K. Lele, Computational aeroacoustics: progress on nonlinear problems of sound generation, Prog. Aerosp. Sci. 40 (2004) 345–416.
[2] S.K. Lele, Compact finite difference schemes with spectral-like resolution, J. Comput. Phys. 103 (1992) 16–42.
[3] D. Gottlieb, S.A. Orszag, Numerical Analysis of Spectral Methods: Theory and Applications, SIAM, 1977.
[4] S. Laizet, E. Lamballais, High-order compact schemes for incompressible flows: a simple and efficient method with quasi-spectral accuracy, J. Comput. Phys. 228 (2009) 5989–6015.
[5] M.P. Simens, J. Jiménez, S. Hoyas, Y. Mizuno, A high-resolution code for turbulent boundary layers, J. Comput. Phys. 228 (2009) 4218–4231.
[6] A.S. Ghate, S.K. Lele, Subfilter-scale enrichment of planetary boundary layer large eddy simulation using discrete Fourier-Gabor modes, J. Fluid Mech. 819 (2017) 494.
[7] A. Uzun, M.R. Malik, Large-eddy simulation of flow over a wall-mounted hump with separation and reattachment, AIAA J. 56 (2018) 715–730.
[8] V. Tritschler, B. Olson, S. Lele, S. Hickel, X. Hu, N.A. Adams, On the Richtmyer–Meshkov instability evolving from a deterministic multimode planar interface, J. Fluid Mech. 755 (2014) 429–462.
[9] J. Ryu, D. Livescu, Turbulence structure behind the shock in canonical shock–vortical turbulence interaction, J. Fluid Mech. 756 (2014).
[10] S. Jagannathan, D.A. Donzis, Reynolds and Mach number scaling in solenoidally-forced compressible turbulence using high-resolution direct numerical simulations, J. Fluid Mech. 789 (2016) 669–707.
[11] B.J. Olson, J. Larsson, S.K. Lele, A.W. Cook, Nonlinear effects in the combined Rayleigh-Taylor/Kelvin-Helmholtz instability, Phys. Fluids 23 (2011) 114107.
[12] D.J. Bodony, S.K. Lele, On using large-eddy simulation for the prediction of noise from cold and heated turbulent jets, Phys. Fluids 17 (2005) 085103.
[13] W.R. Wolf, J.L.F. Azevedo, S.K. Lele, Convective effects and the role of quadrupole sources for aerofoil aeroacoustics, J. Fluid Mech. 708 (2012) 502.
[14] N.S. Ghaisas, A. Subramaniam, S.K. Lele, A unified high-order Eulerian method for continuum simulations of fluid flow and of elastic–plastic deformations in solids, J. Comput. Phys. 371 (2018) 452–482.
[15] J. Shang, High-order compact-difference schemes for time-dependent Maxwell equations, J. Comput. Phys. 153 (1999) 312–333.
[16] S. Nagarajan, S.K. Lele, J.H. Ferziger, A robust high-order compact method for large eddy simulation, J. Comput. Phys. 191 (2003) 392–419.
[17] M.L. Wong, S.K. Lele, High-order localized dissipation weighted compact nonlinear scheme for shock- and interface-capturing in compressible flows, J. Comput. Phys. 339 (2017) 179–209.
[18] A. Subramaniam, M.L. Wong, S.K. Lele, A high-order weighted compact high resolution scheme with boundary closures for compressible turbulent flows with shocks, J. Comput. Phys. 397 (2019) 108822.
[19] W. Gander, G.H. Golub, Cyclic reduction—history and applications, in: Scientific Computing, Hong Kong, 1997, 1997, pp. 73–85.
[20] R.W. Hockney, A fast direct solution of Poisson's equation using Fourier analysis, J. ACM 12 (1965) 95–113.
[21] B.L. Buzbee, G.H. Golub, C.W. Nielson, On direct methods for solving Poisson's equations, SIAM J. Numer. Anal. 7 (1970) 627–656.
[22] O. Buneman, A compact non-iterative Poisson solver, SUIPR report 294, 1969.
[23] R.A. Sweet, A generalized cyclic reduction algorithm, SIAM J. Numer. Anal. 11 (1974) 506–520.
[24] R.A. Sweet, A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension, SIAM J. Numer. Anal. 14 (1977) 706–720.
[25] P.N. Swarztrauber, A direct method for the discrete solution of separable elliptic equations, SIAM J. Numer. Anal. 11 (1974) 1136–1150.
[26] R. Hockney, C. Jesshope, Parallel Computers: Architecture, Programming and Algorithms, Adam Hilger, Bristol, 1981.
[27] Y. Zhang, J. Cohen, J.D. Owens, Fast tridiagonal solvers on the GPU, ACM SIGPLAN Not. 45 (2010) 127–136.
[28] S.P. Hirshman, K.S. Perumalla, V.E. Lynch, R. Sanchez, BCYCLIC: a parallel block tridiagonal matrix cyclic solver, J. Comput. Phys. 229 (2010) 6392–6404.
[29] S.K. Seal, K.S. Perumalla, S.P. Hirshman, Revisiting parallel cyclic reduction and parallel prefix-based algorithms for block tridiagonal systems of equations, J. Parallel Distrib. Comput. 73 (2013) 273–280.
[30] A. Subramaniam, Simulations of Shock Induced Interfacial Instabilities Including Materials with Strength, Stanford University, 2018.
[31] S.S. Vazhkudai, B.R. de Supinski, A.S. Bland, A. Geist, J. Sexton, J. Kahle, C.J. Zimmer, S. Atchley, S. Oral, D.E. Maxwell, et al., The design, deployment, and evaluation of the coral pre-exascale systems, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 661–672.
[32] C.R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D.S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, J. Wilke, Kokkos 3: programming model extensions for the exascale era, IEEE Trans. Parallel Distrib. Syst. 33 (2022) 805–817, https://doi.org/10.1109/TPDS.2021.3097283.
[33] H.C. Edwards, C.R. Trott, D. Sunderland, Kokkos: enabling manycore performance portability through polymorphic memory access patterns, J. Parallel Distrib. Comput. 74 (2014) 3202–3216, https://doi.org/10.1016/j.jpdc.2014.07.003, http://www.sciencedirect.com/science/article/pii/S0743731514001257, Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.
[34] J.R. Bull, A. Jameson, Simulation of the Taylor–Green vortex using high-order flux reconstruction schemes, AIAA J. 53 (2015) 2750–2761.
[35] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G.D. Peterson, R. Roskies, J.R. Scott, N. Wilkins-Diehr, XSEDE: accelerating scientific discovery, Comput. Sci. Eng. 16 (2014) 62–74, https://doi.org/10.1109/MCSE.2014.80.