Future Generation Computer Systems 146 (2023) 260–272

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

TurboStencil: You only compute once for stencil computation

Song Liu [a], Xinhe Wan [a], Zengyuan Zhang [a], Bo Zhao [b], Weiguo Wu [a,*]

[a] School of Computer Science and Technology, Xi'an JiaoTong University, Xi'an, China
[b] School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK

ARTICLE INFO

Article history:
Received 25 September 2022
Received in revised form 28 February 2023
Accepted 17 April 2023
Available online 25 April 2023

Keywords:
Stencil computation
Convolution
Fast Fourier transform
Boundary effect
Data padding
Performance

ABSTRACT

Stencil computation is an important kind of computational mode that widely used in numeric. It iteratively updates the values of the spatial grid points over multiple time steps according to a given pattern. Existing techniques suffer from high complexity of massive iterative computations. Convolution and fast Fourier transform (FFT) provide the possibility to avoid massive iterations and reduce time complexity of stencil computation. However, current convolution-based fast stencil algorithms cannot effectively solve the problems with aperiodic boundary conditions that are common in practical applications. In this paper, we present a novel algorithm, TurboStencil, for linear stencil computations with aperiodic boundary condition. TurboStencil provides a padding method, eliminating the effects of boundary conditions, to enable convolution for all grid points. For symmetric stencil, TurboStencil only computes once by applying FFT, and thus achieves the time complexity of $O(N \log N)$, where N is the grid data size. For asymmetric stencil, TurboStencil also only computes several times by employing a divide-and-conquer method and FFT, and exhibits a lower complexity than existing stencil algorithms. Experimental results demonstrate that TurboStencil outperforms the state-of-the-art convolution-based fast stencil algorithm by up to 777.1×, 43.2×, and 9.4× for symmetric stencil, and 12.9×, 2.0×, and 1.3× for asymmetric stencil, respectively, on 1D, 2D, and 3D benchmarks.

1. Introduction

Stencil computation is a kind of important scientific computational mode, which is widely used in various numerical application fields. It performs the update on a spatial grid following a given pattern, called a *stencil*, over multiple time steps. Specifically, a *stencil* defines the pattern that the value of a spatial grid point at each time step is calculated by the values of its neighboring points at previous time step. A *linear stencil* uses exclusively linear combinations of other points for numerical update. Therefore, stencil computation can effectively solve the partial differential equations of discrete linear systems [1] to simulate the time-varying state of physical models, such as molecular dynamics [2], transient or unsteady heat conduction [3], image processing [4], combustion [5], elastic wave simulation for geophysics [6]. Due to the importance of stencil computations and their massive parallel computing potential, a lot of research work is devoted to improving the execution performance on various computing architectures.

The optimization approaches for stencil computation are mainly divided into three categories, including cache-aware loop tiling methods [7–9], cache-oblivious divide-and-conquer methods [10], and Krylov subspace methods [11,12]. The loop tiling methods [7] use loop transformations to exploit data locality for high level cache hierarchies as well as expose coarse-grained parallelism. The divide-and-conquer methods [13,14] recursively decompose the region into smaller sub-regions to enable cache-oblivious tiling. The divide-and-conquer methods [13] also try to avoid frequent communication between sub-regions to improve the parallelism. These two types of approaches are used to implement direct solvers to obtain exact solutions for stencil problems in a finite number of time steps. On the contrary, the Krylov subspace methods employ mathematical techniques to produce approximations of exact solutions. And they make a trade-off between execution time and accuracy. But the Krylov methods require a high level of expertise for numerical analysis by manual effort, and they only work for a small subset of stencil problems with low dimensional grids. Although all these approaches improve the execution performance to a certain extent, they all explicitly perform massive iterative computations on the points in a spatial grid, and thus the time cost is nontrival.

Recently, a few fast computing methods [15–17] for linear stencil computations are proposed, which uses convolution to generate the final output by evolving the initial data for many time steps at once. These methods directly evolves grid points via skipping the iterative computations legitimately, and thus significantly reduce the time complexity and execution time. And these emerging methods have become the state-of-the-art accelerating

* Corresponding author.
E-mail address: wgwu@xjtu.edu.cn (W. Wu).

https://doi.org/10.1016/j.future.2023.04.019
0167-739X/

techniques for stencil computations. However, such methods are only applicable to stencils with periodic boundary conditions, which means all grid points are evolved by the given stencil. For the aperiodic boundary conditions, the boundary points of the grid are updated following some rules other than the stencil. Therefore, convolution cannot be directly applied to the stencil computations with aperiodic boundary conditions, and the fast computing methods have to handle some points by naïve loop iterative computation. Unfortunately, the processing time of loop computation has been demonstrated much larger than that of convolution [16]. And in many real scientific applications [2,18,19], the stencil problems are often with aperiodic boundary conditions. The inefficiency of the state-of-the-art techniques in processing aperiodic boundary conditions limit the bursts of optimal performance.

In this paper, we propose TurboStencil, a novel convolution-based fast stencil computing algorithm for solving linear systems. As far as we know, TurboStencil is the first algorithm that only computes once for linear stencil computations with both periodic and aperiodic boundary conditions, which almost replaces the naïve iterative computations in the evolution of all grid points. And thus, it greatly reduces the computational time complexity and execution time, while achieving good accuracy and numerical stability.

In summary, this work makes following contributions.

- We present the TurboStencil algorithms to effectively solve the linear stencil problems with constant aperiodic boundary conditions. We provide a novel padding method for eliminating the effect of boundary conditions, to enable convolution for all stencil points. The proposed algorithm almost completely avoids explicit loop iterative computations. It can also be applied to general boundary conditions by performing a small number of iterative computations.
- We implement a one-time computation algorithm for symmetric stencil problems with the computational complexity of $O(N \log N)$ by applying FFT to convolution, where N is the grid data size. We also provide a divide-and-conquer based algorithm for asymmetric stencil problems, recursively performing a pad-convolve-add operation several times, with $O((N^{1/d}+2t)^d \log(N^{1/d}+2t) dT/t)$, where T is the number of time steps, d is the grid dimension and t is a human-chosen parameter. And the algorithm for symmetric stencils theoretically generates solutions for both finite and infinite time steps.
- We experimentally evaluate the computational performance and numerical accuracy of our algorithms over 1D, 2D, and 3D stencil computations. Our algorithms achieve orders of magnitude speedups (up to 777×) for symmetric stencils, compared to a state-of-the-art algorithm, and speedup up to 90× for asymmetric stencils, with comparable loss in accuracy from floating point error.

The rest of this paper is organized as follows. Section 2 presents the problem statement. Section 3 describes TurboStencil for 1D stencil computations. Section 4 generalizes TurboStencil for high dimensional stencil computations. Section 5 shows the experimental results. Section 6 introduces related work, and Section 7 concludes this paper.

2. Problem statement

2.1. Definitions

Stencil computations iteratively updates the value of points of a spatial grid of points $\boldsymbol{D}$ with an initial data $\boldsymbol{A}[0]$ under a *stencil* pattern $\boldsymbol{S}$ for a given finite T time steps, or iteratively update the grid points until the values of points have converged for many time steps. The unspecified iteration time steps for stencil computations to achieve convergent values are referred as "infinite time steps" in this paper. Usually, the *stencil* is described by *n*d*m*pt that the value of a point is updated by its m neighbors, including itself, from previous time step in a n-dimensional grid. The *linear stencil* is an exclusively linear combination of data of neighboring points. For a stencil computation, grid data $\boldsymbol{A}$ and stencil pattern $\boldsymbol{S}$ have the same dimensions and different sizes. A point on the boundaries of the grid is called *boundary point* which follows the boundary conditions for update. Otherwise, it is an *interior point* which performs update by $\boldsymbol{S}$. The data of grid points at time step t is denoted by $\boldsymbol{A}[t]$. Fig. 1 shows the 1d3pt heat stencil codes, and $\boldsymbol{S} = (0.25, 0.5, 0.25)$.

```
for (int t = 0; t < T; t++)
  for (int i = 1; i < N-1; i++)
    A[t+1][i] = 0.25*A[t][i-1] + 0.5*A[t][i] + 0.25*A[t][i+1];
```

Fig. 1. Stencil computation codes of 1d3pt heat equation.

2.2. Boundary conditions

Boundary conditions determine the update rule of boundary points. There are mainly periodic and aperiodic boundary conditions [16]. With periodic boundary condition, boundary points follow the same stencil pattern as interior points, and their update depend on the values of neighboring interior points at previous time step. With aperiodic boundary condition, boundary points follow a different update pattern which does not depend on their neighboring interior points. Generally, the values of these boundary points can be specified by user or calculated by other patterns unrelated to the stencil. In short, the values of all boundary points at all time steps are known with the aperiodic boundary condition before stencil computation.

This paper focuses on the constant aperiodic boundary condition, which is a special case of Dirichlet boundary conditions [20], and is very common in stencil computations [19,21]. With this condition, the values of boundary points remain constant along with time step iterations. For the 1d3pt heat in Fig. 1, constant boundary condition ensures that $\boldsymbol{A}[0][0] = \boldsymbol{A}[1][0] = \cdots = \boldsymbol{A}[T][0]$ and $\boldsymbol{A}[0][N-1] = \boldsymbol{A}[1][N-1] = \cdots = \boldsymbol{A}[T][N-1]$.

2.3. Convolution and boundary effect

With periodic boundary conditions, the data is updated by $\boldsymbol{A}[t+1] = \boldsymbol{S} * \boldsymbol{A}[t]$. And with the associative law of convolution, we have $\boldsymbol{A}[T] = \boldsymbol{S} * \boldsymbol{S} * \cdots * \boldsymbol{S} * \boldsymbol{A}[0] = \boldsymbol{S}^T * \boldsymbol{A}[0]$. The final output can be calculated by convolution of $\boldsymbol{S}$ for T times.

However, with aperiodic boundary conditions, only some points are valid for convolution. Fig. 2 shows the iterative computation process of the 1d3pt heat stencil. Since the boundary points are not applied with *stencil* for update, they affect the update of some *interior points* near the boundary regions (red points) due to the data dependence. These points are invalid to conduct convolution for stencil computation. We call these points as *affected points*, and the effect of boundary points that causes invalid convolution for *affected points* as *boundary effect*. In Fig. 2, only the green *interior points* that are not affected by boundary effect are valid for convolution.

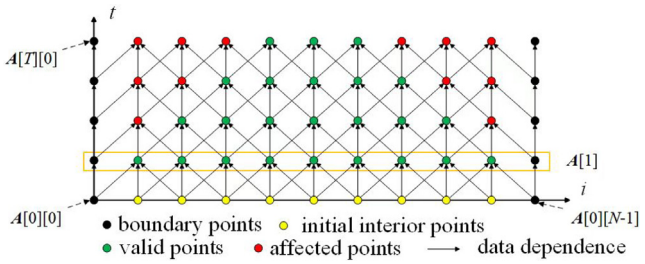


Fig. 2. Iterative computation process of the 1d3pt heat stencil.

2.4. Discrete Fourier transform

Fourier transform (FT) is a basic operation used in digital signal processing to express and analyze time-domain signals. In order to use FT for scientific computing programs, functions must be defined at discrete points rather than in continuous domains, i.e., Discrete Fourier transform (DFT) [22]. According to the **convolution theorem**, circular convolution in the FT forward domain equals multiplication in the FT backward domain. Thus, FT can be used to calculate convolution. There are three steps to calculate convolution $\boldsymbol{U} * \boldsymbol{V}$ with FT: First, perform DFT on $\boldsymbol{U}$ and $\boldsymbol{V}$ to get $\boldsymbol{X_U}$ and $\boldsymbol{X_V}$; second, calculate dot product $\boldsymbol{X_W} = \boldsymbol{X_U} \cdot \boldsymbol{X_V}$; third, perform inverse DFT on $\boldsymbol{X_W}$ to get $\boldsymbol{W}$, and $\boldsymbol{W} = \boldsymbol{U} * \boldsymbol{V}$.

Suppose the size of both $\boldsymbol{U}$ and $\boldsymbol{V}$ is N, the time complexity of DFT is $O(N^2)$, which is the same as that of convolution. There are many methods to implement convolution, such as img2col+GEMM, Winograd [23], and fast Fourier transform (FFT) [22]. The FFT method can reduce the complexity of DFT from $O(N^2)$ to $O(N \log N)$. There are many variants of FFT today, and we employ the FFT implementation from the Intel Math Kernel Library (MKL) [19] to reduce the time complexity and speed up stencil computations. Note that, applying FFT causes accuracy decline [24] due to the floating point errors, but it is still acceptable. The numerical stability is discussed in Section 4.4.

2.5. Motivation

Previous work [16] has noticed the boundary effect but did not eliminate the effect. A fast stencil algorithm [16] uses naïve loop method to iteratively compute the *affected points* and applies fast Fourier transform (FFT) to implement convolution for valid interior points to solve the aperiodic stencil problem. However, the iterative computation takes much more time than FFT and the time complexity of this algorithm is

$O(dTN^{1-1/d} \log(dTN^{1-1/d}) \log T + N \log N)$, where N is the data size, T is time steps, and d is the dimension of spatial grids.

To solve the boundary effect and ensure correct convolution of all grid points, our basic ideas are as follows.

- Points that follow different update patterns should be calculated separately. Because of linearity, the final result is the sum of boundary point result and initial interior point result, generated by evolving the input data of points for many time steps.
- Data padding can be used to eliminate the boundary effect and enable convolution for all points. The padding method should be carefully designed to ensure that the output of convolution is the same as that of naïve iterative computation.

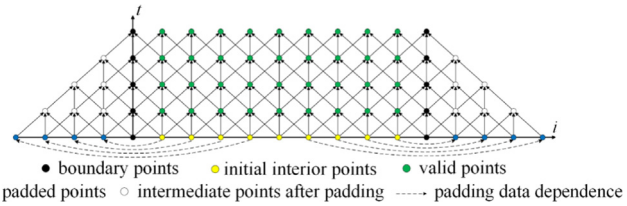


Fig. 3. Illustration of padding method for computing interior points.

3. TurboStencil for 1D stencil

To clearly introduce the proposed TurboStencil algorithm, we use the 1d3pt stencil computation as a walk-through example. Suppose $\boldsymbol{S} = (c_1, c_2, c_3)$, initial input $\boldsymbol{A}[0]$ of grid points $\boldsymbol{D}$ with data size N for the 1d3pt stencil. $\boldsymbol{D}$ consists of $\boldsymbol{D}_i$ and $\boldsymbol{D}_b$. $\boldsymbol{D}_i$ represents interior points and contains points $\boldsymbol{D}[1 : N-2]$. $\boldsymbol{D}_b$ represents boundary points and contains points $\boldsymbol{D}[0]$ and $\boldsymbol{D}[N-1]$. Because of the linearity, the final output is the sum of interior point output and boundary point output over finite time step iterations, i.e., $\boldsymbol{A}[T] = \boldsymbol{I}[T] + \boldsymbol{B}[T]$. $\boldsymbol{A}[T]$ is the final output of $\boldsymbol{D}$, $\boldsymbol{I}[T]$ is the output of $\boldsymbol{D}_i$, and $\boldsymbol{B}[T]$ is the output of $\boldsymbol{D}_b$ over T time steps. Therefore, the stencil computation can be decomposed into computing interior point result and boundary point result.

3.1. Computing interior points

Our idea is to design a padding method to enable all interior points valid for convolution. To exclude the boundary effect, we first set the input values of $\boldsymbol{D}_b$ to 0. Then, we pad $2T$ points to $\boldsymbol{D}$. The number of padded points is determined by T and stencil radius. Here we consider the stencil radius of 1, and the padded $\boldsymbol{D}$ has a size of $N + 2T$. Fig. 3 shows the padding process of interior points for 1d3pt heat. With correct padding values, all interior points, including the *affected points*, can be convolved with $\boldsymbol{S}$ to compute the output.

However, inappropriate padding leads to incorrect results. Thus, the key of padding method is to find the solutions of padded points to ensure that the result of convolution for $\boldsymbol{D}_i$ is correct.

Padding method

The method pads proportional numbers opposite to corresponding input values with the initial two boundary points, respectively, as the symmetric axis, see in Fig. 3. And the input values by padding for interior points are calculated by (1). To verify the correctness of padding, we have to prove that (2) is established based on (1).

$$\boldsymbol{I}[0][i] = \begin{cases} 0 & i = 0 \text{ or } i = N-1 \\ \boldsymbol{A}[0][i] & 0 < i < N-1 \\ -\boldsymbol{A}[0][-i] \cdot (c_3/c_1)^i & -T \le i < 0 \\ -\boldsymbol{A}[0][2N-2-i] \cdot (c_3/c_1)^{-i} & N-1 < i \le N-1+T \end{cases} \tag{1}$$

$$\boldsymbol{I}[T] = \boldsymbol{S}^T * \boldsymbol{I}[0] \tag{2}$$

Proof. For the affected points next to left boundary point $\boldsymbol{D}[t][0]$, the output $\boldsymbol{I}[t][i]$ of these points at t time step is written as (3),

$$\boldsymbol{I}[t][i] = \boldsymbol{S}^t \cdot \boldsymbol{I}[0][i-t : i+t] - \sum_{j=0}^{t-i} e_{ij} \boldsymbol{I}[j][0], \ 0 \le i \le t \tag{3}$$

where the dot product is the output evolved with related input values (including initial values and padded values), the summation is the computational deviation of the output caused by

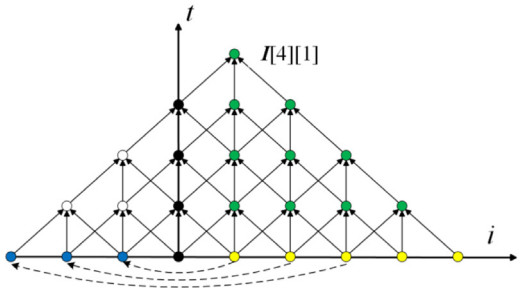


Fig. 4. Illustration of the computational deviation deriving from boundary points after padding.

the padding points, and the e_{ij} is related to $\boldsymbol{S}$ and is the coefficient of the linear representation of the deviation. Fig. 4 gives an illustration of the deviation. The values of boundary points evolve with the padded data, while they should have remained at 0. And thus, a computational deviation is introduced to the value of $\boldsymbol{I}[4][1]$. The output of $\boldsymbol{I}[4][1]$ needs to subtract the extra computing results from boundary points.

Applying the binomial theorem to the given $\boldsymbol{S} = (c_1, c_2, c_3)$, we obtain (4), where *index* is the index of element of $\boldsymbol{S}^t$.

$$\boldsymbol{S}^t[index] = \sum_{i+j=0}^{t} \frac{t!}{i!j!k!} c_1^i c_2^j c_3^k \tag{4}$$
$$(i + j + k = t, k - i = index, -t \le index \le t)$$

Hence, we derive (5) from (4).

$$\boldsymbol{S}^t[-index] = \boldsymbol{S}^t[index] \cdot (c_1/c_3)^{index} \tag{5}$$

Then, we can deduce (6).

$$\begin{aligned}\boldsymbol{I}[j][0] =& \boldsymbol{S}^t \cdot \boldsymbol{I}[0][-index : index] \\ =& \sum_{y=-index}^{index} \boldsymbol{S}^t[y]\boldsymbol{I}[0][y] \\ =& \boldsymbol{S}^t[0]\boldsymbol{I}[0][0] + \sum_{y=1}^{index}(\boldsymbol{S}^t[y]\boldsymbol{I}[0][y] + \boldsymbol{S}^t[-y]\boldsymbol{I}[0][-y]) \\ =& 0 + \sum_{y=1}^{index}(\boldsymbol{S}^t[y]\boldsymbol{I}[0][y] + \boldsymbol{S}^t[y] \cdot (c_1/c_3)^y \cdot \boldsymbol{I}[0][y] \cdot (-(c_3/c_1)^y)) \\ =& 0\end{aligned} \tag{6}$$

Next, we bring (6) into (3) and obtain (7).

$$\boldsymbol{I}[t][i] = \boldsymbol{S}^t \cdot \boldsymbol{I}[0][i - t : i + t] \tag{7}$$

For the affected points next to right boundary point $\boldsymbol{D}[t][N-1]$, the proof process is the same, and they also satisfy (7). Therefore, $\boldsymbol{I}[T] = \boldsymbol{S}^T * \boldsymbol{I}[0]$ is proved. □

According to the padding method, the interior point result can be obtained by computing convolution $\boldsymbol{I}[T] = \boldsymbol{S}^T * \boldsymbol{I}[0]$. However, there is a limitation for padding that $T \le N - 2$. If $T > N - 2$, points are out the scope of $\boldsymbol{I}[0][1 - N : 2N - 2]$, and cannot be padded. However, this limitation can be solved by our algorithms, which is described in Sections 3.4 and 3.5.

3.2. Computing boundary points

With constant boundary conditions, boundary points remain the input values during time step iterations and have boundary effect on affected points. To exclude the effect of interior points on computing boundary point result, we first set the values of initial interior points to 0 and design the input value of $\boldsymbol{B}[0][i]$ by (8).

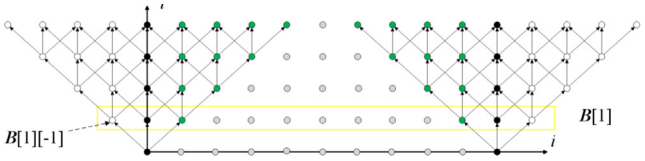


Fig. 5. Process of computing boundary points.

$$\boldsymbol{B}[0][i] = \begin{cases} \boldsymbol{A}[0][0] & i = 0 \\ -\boldsymbol{A}[0][N-1] & i = N-1 \\ 0 & 1 \le i \le N-2 \end{cases} \tag{8}$$

Fig. 5 shows the process of computing boundary points. For all left boundary points $\boldsymbol{B}[t][0]$ at t time step, they follow the stencil pattern $(c_1, 0, -c_3)$, while all right boundary points $\boldsymbol{B}[t][N-1]$ follow the stencil pattern $(-c_1, 0, c_3)$. For simplicity, we use stencil $\boldsymbol{S}_1 = (c_1, 0, -c_3)$ to compute all boundary point results, and thus the input value of $\boldsymbol{B}[0][N-1]$ is opposite to initial value in (8). Therefore, $\boldsymbol{B}[1] = \boldsymbol{S}_1 * \boldsymbol{B}[0]$, and $\boldsymbol{B}[1]$ contains correct result from $\boldsymbol{B}[0]$ and intermediate values of $\boldsymbol{B}[1][-1]$ and $\boldsymbol{B}[1][N]$. With $\boldsymbol{B}[1]$, $\boldsymbol{B}[t+1]$ can be derived from $\boldsymbol{B}[t]$ recursively for each iteration. To solve $\boldsymbol{B}[t+1]$, $\boldsymbol{B}[t]$ is divided into interior points and boundary points. And $\boldsymbol{B}[t+1]$ is the sum of the interior point result obtained by convolving $\boldsymbol{B}[t]$ with given $\boldsymbol{S}$ and the boundary result $\boldsymbol{B}[1]$. Formally, the recurrence formula is described by (9).

$$\boldsymbol{B}[t+1] = \boldsymbol{S} * \boldsymbol{B}[t] + \boldsymbol{B}[1] \tag{9}$$

3.3. Applying FFT to convolution

The size of $\boldsymbol{A}[0]$ and $\boldsymbol{S}$ should be the same for the dot product operation in applying FFT to convolution. Since the input data $\boldsymbol{A}[0]$ has been padded with corresponding values according to the padding method, $\boldsymbol{S}$ also should be padded to make the size is consistent with that of the padded input data. In order to make the padded data not affect the correctness of the results, we pad $\boldsymbol{S}$ with zeros, and use all points of padded $\boldsymbol{A}[0]$ and $\boldsymbol{S}$ to perform FFT computation. The computing result of dot product in the FT backward domain will contain some useless padded data, which will be discarded to obtain the final result. In addition, before performing FFT for convolution, $\boldsymbol{S}$ is rotated to the left, making the central element is in the first, to align with the padded input data, so that the order of output vector after FFT is consistent with final output. Besides, $\boldsymbol{S}$ should be padded with zeros to make the length of padded $\boldsymbol{S}$ is consistent with that of padded input data for FFT.

When applying FFT to (2) and (9), we obtain the output results of $\boldsymbol{I}[T]$ and $\boldsymbol{B}[T]$ in the FT backward domain by (10) and (11), respectively. $\boldsymbol{X}$ denotes the result of FT. In (10) and (11), $\boldsymbol{X}_{\boldsymbol{I}}[0]$, $\boldsymbol{X}_{\boldsymbol{S}}$ and $\boldsymbol{X}_{\boldsymbol{B}}[1]$ are both constant vectors. So, $\boldsymbol{X}_{\boldsymbol{B}}[t+1][i]$ only depends on $\boldsymbol{X}_{\boldsymbol{B}}[t][i]$, and the data dependence has changed after Fourier transform. And the $\boldsymbol{X}_{\boldsymbol{B}}[T][i]$ only depends on $\boldsymbol{X}_{\boldsymbol{B}}[1][i]$. Then, the general formula for $\boldsymbol{X}_{\boldsymbol{B}}[T][i]$ is given by (12). And the output of $\boldsymbol{A}[T]$ in the FT backward domain is given by (13), in where $\boldsymbol{B}[1] = \boldsymbol{S}_1 * \boldsymbol{B}[0]$. Then, the inverse FFT is performed on $\boldsymbol{X}_{\boldsymbol{A}}[T]$, and the unnecessary padded data of the result is discarded to generate the final output $\boldsymbol{A}[T]$.

$$\boldsymbol{X_I}[T][i] = \boldsymbol{X_I}[0][i]\boldsymbol{X_S}[i]^T \tag{10}$$

$$\boldsymbol{X_B}[t+1][i] = \boldsymbol{X_B}[t][i]\boldsymbol{X_S}[i] + \boldsymbol{X_B}[1][i] \tag{11}$$

$$\boldsymbol{X_B}[T][i] = \begin{cases} T\boldsymbol{X_B}[1][i] & \boldsymbol{X_S}[i] = 1 \\ \boldsymbol{X_B}[1][i]\frac{1-\boldsymbol{X_S}[i]^T}{1-\boldsymbol{X_S}[i]} & \boldsymbol{X_S}[i] \neq 1 \end{cases} \tag{12}$$

$$\boldsymbol{X_A}[T][i] = \begin{cases} \boldsymbol{X_I}[0][i] + T\boldsymbol{X_B}[1][i] & \boldsymbol{X_S}[i] = 1 \\ \boldsymbol{X_I}[0][i]\boldsymbol{X_S}[i]^T + \boldsymbol{X_B}[1][i]\frac{1-\boldsymbol{X_S}[i]^T}{1-\boldsymbol{X_S}[i]} & \boldsymbol{X_S}[i] \neq 1 \end{cases} \tag{13}$$

There could be a convergence in (13) while $T \to \infty$, which provides the possibility to solve the convergence value problem directly. However, there are some problems about numerical stability because of the floating-point error. We will discuss them in Section 4.4.

3.4. TurboStencil for symmetric stencil

A symmetric stencil is like $\boldsymbol{S} = (c_1, c_2, c_1)$ for 1d3pt or $\boldsymbol{S} = \begin{bmatrix} c_1 & c_2 & c_1 \\ c_3 & c_4 & c_3 \\ c_1 & c_2 & c_1 \end{bmatrix}$ for 2d9pt, that elements are symmetrical with respect to the central element in $\boldsymbol{S}$. This is very common in stencil computation, such as Heat, Jacobi, Poisson, etc.

The symmetry brings great benefits for our approach. First, it eliminates the need to compute powers when padding, since powers have a base of 1. Second, and more importantly, it allows the input to be padded to a circular vector and breaks the limitation that $T \leq N-2$. For example, given that $\boldsymbol{I}[0] = [0\ 1\ 2\ 3\ 0]$, and the padded initial input is $[0\ 1\ 2\ 3\ 0\ -3\ -2\ -1\ 0\ 1\ 2\ 3\ 0\ -3\ -2\ -1\ 0\ 1\ 2\ 3\ 0]$ for T=8 according to the padding method, which is a circular vector with a period of $[0\ 1\ 2\ 3\ 0\ -3\ -2\ -1]$. Then, only one period of the circular vector is stored in memory, and the input data that needs to be stored is reduced from $N+2T$ to $2N-2$, as T is generally much larger than N. Based thereon, we can perform circular convolution and FFT on symmetric stencil computation. And the FFT is very suitable for large scale circular convolution and reduces the time complexity from $O(N^2)$ to $O(N \log N)$.

Now, we introduce the TurboStencil algorithm for 1D symmetric stencil computation. For the given input vector $\boldsymbol{A}[0]$ with the size of N, stencil pattern $\boldsymbol{S}$, and time steps T, $\boldsymbol{A}_0[0]$ and $\boldsymbol{A}_1[0]$ which respectively denote the input values of interior points and boundary points are derived from $\boldsymbol{A}[0]$, and $\boldsymbol{S}_0$ and $\boldsymbol{S}_1$ which respectively denote the stencil patterns of interior points and boundary points are derived from $\boldsymbol{S}$.

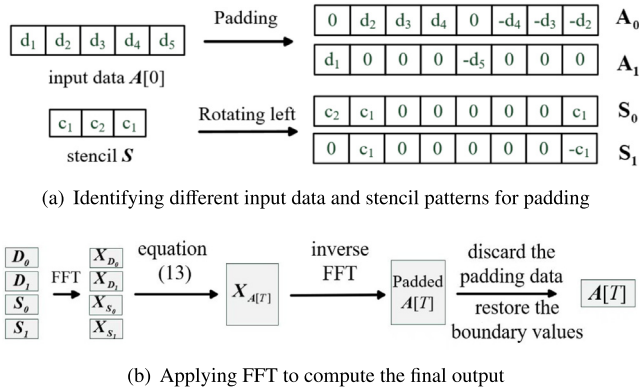
According to the padding method for symmetric stencil described above, padded input values of $\boldsymbol{A}_0[0]$ and $\boldsymbol{A}_1[0]$ with the size of $2N-2$ are given by (14) and (15). Before performing circular convolution, $\boldsymbol{S}_0$ and $\boldsymbol{S}_1$ are rotated to the left and padded with zeros by (16).

$$\boldsymbol{A}_0[i] = \begin{cases} 0 & i = 0 \text{ or } i = N-1 \\ \boldsymbol{A}[0][i] & 0 < i < N-1 \\ -\boldsymbol{A}[0][2N-2-i] & N-1 < i < 2N-2 \end{cases} \tag{14}$$

$$\boldsymbol{A}_1[i] = \begin{cases} \boldsymbol{A}[0][0] & i = 0 \\ -\boldsymbol{A}[0][N-1] & i = N-1 \\ 0 & 0 < i < N-1 \text{ or } N-1 < i < 2N-2 \end{cases} \tag{15}$$

$$\boldsymbol{S}_0 = (c_2, c_1, 0, 0, \ldots, c_1),\ \boldsymbol{S}_1 = (0, c_1, 0, 0, \ldots, -c_1) \tag{16}$$

Then, we perform FFT on these four vectors in parallel and compute the middle results in FT backward domain by (13).



(a) Identifying different input data and stencil patterns for padding

(b) Applying FFT to compute the final output

Fig. 6. Process of TurboStencil for 1D symmetric stencil.

Next, we perform inverse FFT on the middle results, and discard padding data and restore boundary values to generate the final output. Fig. 6 shows the process of TurboStencil for 1D symmetric stencil computation.

3.5. TurboStencil for asymmetric stencil

For asymmetric stencils, elements are not symmetrical with respect to the central element in $\boldsymbol{S}$. And there is a limitation that $T \leq N-2$ for directly applying the padding method. For the cases that $T > N-2$, we employ a divide-and-conquer method to break the limitation.

Our method is to choose a proper time step band t to make $t \leq N-2$, and then we perform a pad-convolve-add operation to compute the t iterations. For each t time step band, we pad the input data, and convolve $\boldsymbol{S}^t$ with input data to respectively compute the interior point result and the boundary point result by FFT, and then add the results to get the output over t iterations according to the methods described in Sections 3.1 to 3.3. And the output is used as the input data for the next t time step band. This process is repeated for T/t times to generate the final output over T iterations. If T cannot be divided by t, the rest iterations are computed by naïve loop method. This case will increase the computation time, and thus it should be avoided as much as possible.

Fig. 7 shows the process of TurboStencil for 1D asymmetric stencil computation. Here, the linear convolution is used. And some intermediate results, such as $\boldsymbol{X_S}[i]^T$ and $\boldsymbol{X_B}[t][i]$, can be reused and only need to be calculated once. The selection of time step band t has an influence on the performance of our method, and it is discussed in Section 4.2.

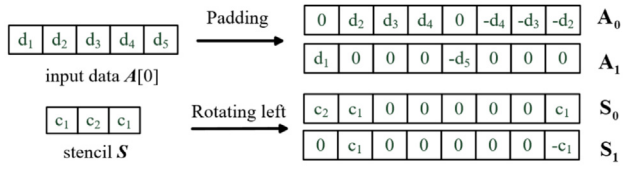
4. Generalization and implementation of TurboStencil

Based on the core ideas of TurboStencil, it can be extended for higher-dimensional stencil computations. We use 2D case to demonstrate the problems and process for extension, and there is no essential difference for 3D or higher-dimensional cases. Based thereon, we propose the general TurboStencil algorithms.

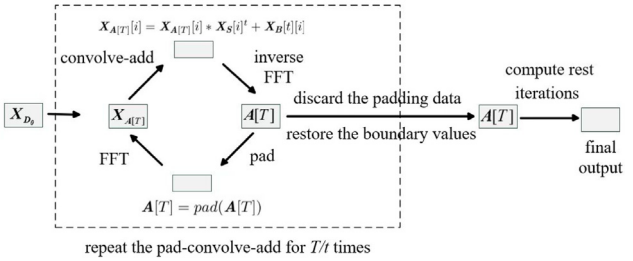
4.1. Main problems for extension

For 2D case, there are different types of boundary points. Correctly identifying boundary effects and padding for different input values are the main problems for extension.

Different boundary effects. Suppose a 2d9pt stencil with an input data of $\boldsymbol{A}[0]$ ($\boldsymbol{A}[0]$ is a matrix) and a stencil pattern $\boldsymbol{S}$ =



(a) Identifying different input data and stencil patterns for padding



(b) Performing pad-convolve-add operation repeatedly to compute the final output

Fig. 7. Process of TurboStencil for 1D asymmetric stencil.

$\begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix}$. There are four types of points and corresponding stencil patterns for 2D stencil, see in Fig. 8. And three of them are related to boundary points. An interior point affects its surrounding 8 neighbors and itself in one iteration. A point on the up/down boundary affects its 3 down/up neighbors, and a point on the left/right boundary affects its 3 right/left neighbors. A vertex affects one point on its diagonal.

We reassert our idea that points that follow different stencil patterns should be calculated separately. So, we derive four patterns, i.e., $\boldsymbol{S}_0$, $\boldsymbol{S}_1$, $\boldsymbol{S}_2$ and $\boldsymbol{S}_3$, from $\boldsymbol{S}$ to describe the stencils of interior points, up/down boundary points, left/right boundary points, and vertexes, respectively. Fig. 8 gives the specific forms of these stencil patterns.

Correct padding. To apply the padding method, the opposite rows or columns of $\boldsymbol{S}$ should be proportional. Formally, $\exists k_1 \neq 0, [c_{00}\ c_{10}\ c_{20}] = k_1[c_{02}\ c_{12}\ c_{22}]$ and $\exists k_2 \neq 0, [c_{00}\ c_{01}\ c_{02}] = k_2[c_{20}\ c_{21}\ c_{22}]$. If $\boldsymbol{S}$ is not accord with this rule, no padding method could give correct results. However, common stencils often meet this rule, and most of them are symmetric.

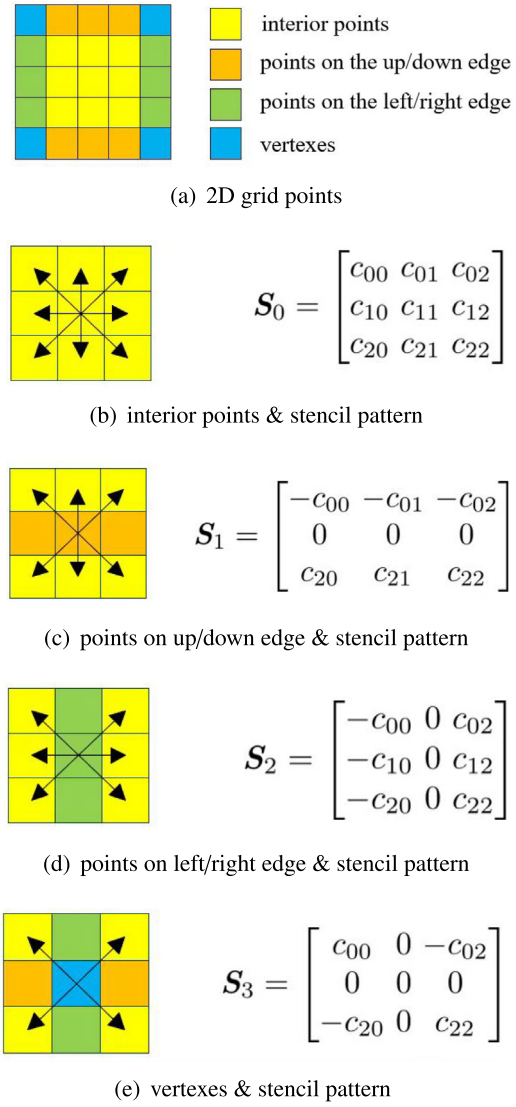
The input data $\boldsymbol{A}[0]$ is divided into initial values of interior points, up/down boundary points, left/right boundary points, and vertexes, denoted by $\boldsymbol{A}_0$, $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, and $\boldsymbol{A}_3$, respectively.

For the interior points, each row and column of $\boldsymbol{A}_0$ is padded in the same way as 1D case, but replacing the c_1/c_3 with k_1 and k_2 in (1) for row and column data, respectively. For the boundary points, $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, and $\boldsymbol{A}_3$ are padded by (8), and the padded values should follow the corresponding stencil patterns to determine that they are opposite number of the boundary point initial value or not.

Fig. 9 shows the padding process for the given 2d9pt stencil. To perform FFT for convolution, $\boldsymbol{S}_0$, $\boldsymbol{S}_1$, $\boldsymbol{S}_2$, and $\boldsymbol{S}_3$ need to be cyclic left and up shifted and padded with zeros. Then, (13) can be naturally extended to 2D case by (17) to apply FFT computing the output,

$$\boldsymbol{X}_{\boldsymbol{A}}[T][i][j] = \begin{cases} \boldsymbol{X}_{\boldsymbol{A}_0}[i][j] + T\boldsymbol{X}_{\boldsymbol{B}}[i][j] & \boldsymbol{X}_{\boldsymbol{S}}[i][j] = 1 \\ \boldsymbol{X}_{\boldsymbol{A}_0}[i][j]\boldsymbol{X}_{\boldsymbol{S}}[i][j]^T + \boldsymbol{X}_{\boldsymbol{B}}[i][j]\frac{1-\boldsymbol{X}_{\boldsymbol{S}}[i][j]^T}{1-\boldsymbol{X}_{\boldsymbol{S}}[i][j]} & \boldsymbol{X}_{\boldsymbol{S}}[i][j] \neq 1 \end{cases} \tag{17}$$

where $\boldsymbol{X}_{\boldsymbol{B}}$ represents the sum three types of boundary point results in one iteration, and $\boldsymbol{X}_{\boldsymbol{B}}[i][j] = \boldsymbol{X}_{\boldsymbol{A}_1}[i][j]\boldsymbol{X}_{\boldsymbol{S}_1}[i][j] +$



(a) 2D grid points

(b) interior points & stencil pattern

(c) points on up/down edge & stencil pattern

(d) points on left/right edge & stencil pattern

(e) vertexes & stencil pattern

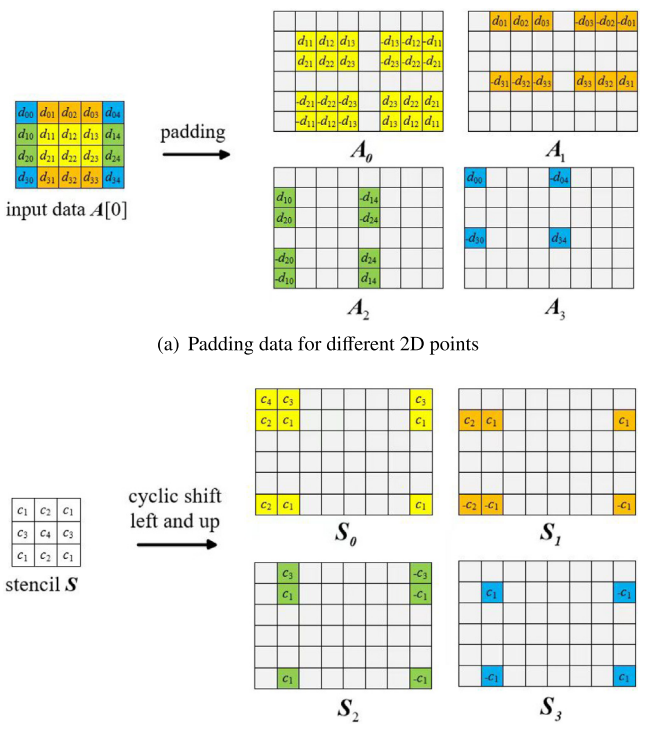
Fig. 8. Four different types of points and stencil patterns in 2D case.

$\boldsymbol{X}_{\boldsymbol{A}_2}[i][j]\boldsymbol{X}_{\boldsymbol{S}_2}[i][j] + \boldsymbol{X}_{\boldsymbol{A}_3}[i][j]\boldsymbol{X}_{\boldsymbol{S}_3}[i][j]$. And the final output is obtained by performing an inverse FFT on the output in FT backward domain, discarding padding data and restoring boundary values.

4.2. General TurboStencil algorithms for symmetric and asymmetric stencil

When we identify different types of boundary points and use correct padding method generating padded input data and stencil pattern matrices, the circular convolution method for 1D symmetric stencil and the pad-convolve-add operation-based divide-and-conquer method for 1D asymmetric stencil can be directly used for 2D cases. And we can generalize the TurboStencil algorithms to the general case.

In a general d-dimensional stencil, there are 2^d types of different points and stencil patterns. Then, we generate stencil patterns, i.e., $\boldsymbol{S}_0$, $\boldsymbol{S}_1$, …, and $\boldsymbol{S}_{2^d-1}$, from given $\boldsymbol{S}$, and divide input data $\boldsymbol{A}$ into $\boldsymbol{A}_0$, $\boldsymbol{A}_1$, …, and $\boldsymbol{A}_{2^d-1}$. The padding on each of the $\boldsymbol{A}$ into $\boldsymbol{A}_0$, $\boldsymbol{A}_1$, …, and $\boldsymbol{A}_{2^d-1}$ should be in d dimensions independently. And $\boldsymbol{S}_0$, $\boldsymbol{S}_1$, …, and $\boldsymbol{S}_{2^d-1}$ need to be cyclic shifted along d dimensions and padded with zeros for applying FFT to convolution. The output in FT backward domain is the sum of all



(a) Padding data for different 2D points

(b) Deriving different stencil patterns from S and padding them

Fig. 9. An illustration of padding data and identifying stencil patterns for 2D points. Colored grids represent the input values of different types of points, and same colored points correspond to the same colored stencil pattern. Gray grids represent the padded data, and their values are 0.

Algorithm 1 TurboStencil for d-dimensional symmetrical linear stencil computation

Require: input data $\boldsymbol{A}[0]$ with size N, stencil pattern $\boldsymbol{S}$, time steps T

Ensure: $\boldsymbol{A}[T]$

1: generate $\boldsymbol{S}_0, \boldsymbol{S}_1, \cdots, \boldsymbol{S}_{2^d-1}$ from $\boldsymbol{S}$ and perform cyclic shift and pad them with zeros;
2: generate $\boldsymbol{A}_0, \boldsymbol{A}_1, \cdots, \boldsymbol{A}_{2^d-1}$ from $\boldsymbol{A}[0]$ and pad on them;
3: do FFT on $\boldsymbol{S}_0, \boldsymbol{S}_1, \cdots, \boldsymbol{S}_{2^d-1}$ and $\boldsymbol{A}_0, \boldsymbol{A}_1, ..., \boldsymbol{A}_{2^d-1}$ in parallel;
4: compute output of $\boldsymbol{X}_{\boldsymbol{A}[T]}$ by (13); /* Here $\boldsymbol{X}_{\boldsymbol{B}} = \boldsymbol{X}_{\boldsymbol{A}_1} \cdot \boldsymbol{X}_{\boldsymbol{S}_1} + \boldsymbol{X}_{\boldsymbol{A}_2} \cdot \boldsymbol{X}_{\boldsymbol{S}_2} + \cdots + \boldsymbol{X}_{\boldsymbol{A}_{2^d-1}} \cdot \boldsymbol{X}_{\boldsymbol{S}_{2^d-1}}$ */
5: do inverse FFT on $\boldsymbol{X}_{\boldsymbol{A}[T]}$;
6: discard the padding data and restore the boundary values;
7: return $\boldsymbol{A}[T]$;

2^d different results. Other than that, the computation process for d-dimensional stencil is the same as the 1D and 2D cases.

Algorithm 1 and 2 give the TurboStencil for d-dimensional symmetric and asymmetric linear stencil computation, respectively. The time complexity of TurboStencil is $O(N \log N)$ for symmetric stencil and $O((N^{1/d} + 2t)^d \log(N^{1/d} + 2t) dT/t)$ for asymmetric stencil, and t is the height of time step band, which is a human-chosen parameter.

The selection of t time step band has an influence on the performance of asymmetric stencil. To be specific, a larger value of t leads to more cost of once convolution but less numbers of convolutions, and vice versa. The cost of padding is mainly related to the memory access performance, while the cost of FFT is related to both memory access and computing performance.

Algorithm 2 TurboStencil for d-dimensional asymmetrical linear stencil computation

Require: input data $\boldsymbol{A}[0]$ with size N, stencil pattern $\boldsymbol{S}$, time steps T, height of time step band t

Ensure: $\boldsymbol{A}[T]$

1: generate $\boldsymbol{S}_0, \boldsymbol{S}_1, \cdots, \boldsymbol{S}_{2^d-1}$ from $\boldsymbol{S}$ and perform cyclic shift and pad them with zeros;
2: generate $\boldsymbol{A}_0, \boldsymbol{A}_1, \cdots, \boldsymbol{A}_{2^d-1}$ from $\boldsymbol{A}[0]$ and pad on them;
3: do FFT on $\boldsymbol{S}_0, \boldsymbol{S}_1, \cdots, \boldsymbol{S}_{2^d-1}$ and $\boldsymbol{A}_0, \boldsymbol{A}_1, ..., \boldsymbol{A}_{2^d-1}$ in parallel;
4: compute t iterations' boundary effect $\boldsymbol{X}_{\boldsymbol{B}}[t][i]$ by (13) and convolution kernel $\boldsymbol{X}_{\boldsymbol{S}}[i]^t$ in FT backward domain;
5: **for** $itr = 0 \rightarrow T/t$ **do**
6: $\quad \boldsymbol{A}[T] = pad(\boldsymbol{A}[T])$; /*pad operation*/
7: $\quad \boldsymbol{X}_{\boldsymbol{A}[T]} = FFT(\boldsymbol{A}[T])$;
8: $\quad$ **for** $i = 0 \rightarrow (N^{1/d} + 2t)^d - 1$ **do**
9: $\quad\quad \boldsymbol{X}_{\boldsymbol{A}[T]}[i] = \boldsymbol{X}_{\boldsymbol{A}[T]}[i] * \boldsymbol{X}_{\boldsymbol{S}}[i]^t + \boldsymbol{X}_{\boldsymbol{B}}[t][i]$; /*convolve-add operation*/
10: $\quad$ **end for**
11: $\quad \boldsymbol{A}[T] = iFFT(\boldsymbol{X}_{\boldsymbol{A}[T]})$;
12: **end for**
13: discard the padding data and restore the boundary values;
14: compute rest iterations in loop method;
15: return $\boldsymbol{A}[T]$;

Therefore, the best selection of t is related to hardware architectures and is difficult to model accurately. And we chose the value of t empirically in this paper.

4.3. General aperiodic boundary condition problem

Although TurboStencil is specially designed for stencil computations with the constant aperiodic boundary condition, it can also be applied to more general aperiodic boundary conditions that the values of boundary points are not constant at each time step, i.e., the Dirichlet boundary condition [20]. Since TurboStencil calculates the results of interior points and boundary points separately, the computing of interior points based on the padding method is the same. We only need to consider the computing of boundary points and affected points.

Fig. 10 shows the illustration of our algorithm applied to one-dimensional stencil computation with general aperiodic boundary condition. The boundary points $\boldsymbol{A}[0{:}t][0]$ and $\boldsymbol{A}[0{:}t][N\text{-}1]$ will affect the results of points $\boldsymbol{A}[t][1{:}t]$ and $\boldsymbol{A}[t][N\text{-}1\text{-}t{:}N\text{-}1]$, respectively, after t time steps. As the values of boundary points at each time step is not constant, the calculation of (12) is no longer applicable to the results of boundary points. We use the naïve loop computation method to calculate the results of boundary points and affected points. In this case, for both symmetric and asymmetric stencils, we employ the divide-and-conquer method used in TurboStencil for asymmetric stencil to split the time steps into several time step bands. Then, we separately calculate the results of boundary points and affected points (i.e., the red triangle regions) by the naïve loop computation method and add the results with the results of interior points within each time step band. Only the add operation in each pad-convolve-add is different from the TurboStencil for asymmetric stencil, and other processing steps are the same, that is, the values of interior points (i.e., the green trapezoidal regions) are calculated by the padding and convolution methods. The naïve loop computation of red triangle regions can be performed in parallel and the results are obtained before the iterative processes of pad-convolve-add operations.

Although the naïve loop computation method can be directly performed on all boundary points and affected points (i.e., the

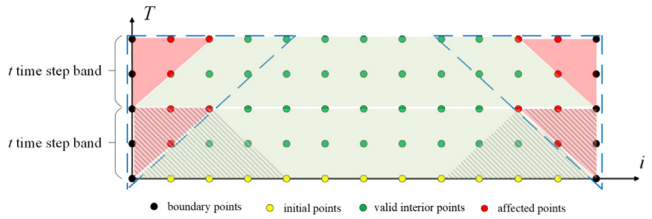


Fig. 10. Process of divide-and-conquer method for general aperiodic boundary condition.

regions within the blue triangle) to compute the results of output for symmetric stencil, this will lead to a massive number of iterative computations and lower the computational performance of the algorithm. The divide-and-conquer method can effectively reduce the time-consuming iterative computations that only a small portion of points are calculates in the naïve loop iterative way. The state-of-the-art fast stencil algorithm [16] also uses the naïve loop computation method to handle the aperiodic boundary condition problem, which needs to iteratively compute a large number of points (i.e., the shadowed small trapezoidal regions) in each time step band, while our algorithm can reduce the amounts of iterative computations by 2/3.

4.4. Discussion

Numerical stability

Theoretically, our algorithm should have the same results of the naïve loop method. But FFT and complex number calculations accumulate and exaggerate the floating-point errors, and may cause the numerical instability of the algorithm.

Let us review (13). $\boldsymbol{X_A}[T][i]$ and $\boldsymbol{X_S}[i]$ are complex numbers. Because of the floating-point error, whether the $\boldsymbol{X_S}[i]$ equals 1 should be judged carefully. If it is misjudged, the algorithm will give wrong results. A commonly used method is to set an upper error limit. This method does work but needs manual labors to adjust the parameter. However, identifying which $\boldsymbol{X_S}[i]$ equals 1 before doing FFT is a potential way to solve the problem of misjudgment, and we are going to study in further work.

For the stencil problems that compute convergence values for infinite time steps, $T \to \infty$, we can use (18) instead of (13), with the conditions: (a)$|\boldsymbol{X_S}[i]| < 1$ or (b)$\boldsymbol{X_S}[i] = 1$ and $\boldsymbol{X_B}[i] = 0$, to compute the output. And these two conditions are very common in most well-designed stencils.

$$\boldsymbol{X_A}[T][i] = \begin{cases} \boldsymbol{X_I}[i] & \boldsymbol{X_S}[i] = 1 \\ \boldsymbol{X_B}[1][i]\frac{1-\boldsymbol{X_S}[i]^T}{1-\boldsymbol{X_S}[i]} & \boldsymbol{X_S}[i] \neq 1 \end{cases} \tag{18}$$

Applicability

(a) Linearity. Our algorithms are applicable to linear stencil computations for solving the discrete linear systems. Linear stencils are quite common in computational numeric methods [25,26]. Loop optimization-base approaches are applicable to nonlinear stencils, but they suffer high computational complexity. To the best of our knowledge, there are three convolution-based approaches for stencil computations [15–17], and they are inapplicable to nonlinear stencils.

(b) Boundary conditions. Our algorithms are effective for the aperiodic boundary conditions, specifically the constant boundary conditions. The state-of-the-art FFT-based method [16] focuses on solving the stencil problems with periodic boundary conditions. But it fails to effectively handle with the aperiodic boundary conditions. Our algorithms can also be applied to general aperiodic boundary conditions with introducing a small number of loop iterative computations.

(c) Stencil pattern. To apply our algorithms, the stencil pattern should be proportional. The symmetric stencil is a special case of proportional stencil. The convolution-based methods [15,17] are only applicable to symmetric stencils while ours to both symmetric and asymmetric stencils. And the FFT-based method [16] can solve proportional and disproportional stencils.

(d) Finite or infinite time steps. Our algorithm for symmetric stencils is applicable to both cases, while the asymmetric algorithms is applicable to finite time steps. The loop-based approaches cannot be applied to stencils with infinite time steps, as well as some convolution-based methods [15,17]. And the FFT-based method [16] cannot solve the stencil problems for infinite time steps with the aperiodic boundary conditions.

Memory cost

TurboStencil is to exchange memory space for fast computing time. The use of FFT requires more costs of memory, i.e., it stores one copy of real numbers and two copies of complex numbers. The memory requirement of applying FFT is three times that of the input data. However, as the fast DFT algorithm, FFT is still widely used to quickly solve some large-scale industrial and scientific problems, such as digital image processing, modern radar data analysis, data acquisition, and locomotive fault detection. Since TurboStencil derives different points and stencil patterns to separately perform FFT based the proposed padding method, it also brings additional memory overhead. We will specifically analyze the memory cost of the proposed algorithms.

For a d-dimensional input data of grid points with N data size, the naïve loop iterative computation method requires $2N$ memory space and exhibits the time complexity of $O(NT)$, where T is the time steps. For the symmetric stencil computation, in each dimension of points, our algorithm respectively generates $(2N^{1/d} - 2)$ padded data of interior points and boundary points, requiring a $2 \times (2N^{1/d} - 2)$ memory space. It also needs to derive different stencil patterns and pad them, and then performs FFT on different points, separately. Therefore, TurboStencil requires total $(2 \times (2N^{1/d} - 2))^d \times 2 \times 3$ memory space for symmetric stencil computation, and achieves the time complexity of $O(N \log N)$. The memory space is less than $(2 \times (2N^{1/d}))^d \times 2 \times 3$, i.e., the memory requirement of TurboStencil is less than 3×4^d times that of the naïve loop iterative computation method. Generally, stencil computations have 2D or 3D grid points, and T is more than thousands to millions, or even more. In other words, TurboStencil has gained thousands of times of improvement in computing time with less than 200 times of memory overhead, which is cost-effective. Similarly, for the asymmetric stencil computation, TurboStencil requires total $(2 \times (N^{1/d} + 2t))^d \times 2 \times 3$ memory space, and achieves the time complexity of $O((N^{1/d} + 2t)^d \log(N^{1/d} + 2t)^d T/t)$, where t is the manually tuned height of time step band. Compared with the naïve loop computation method, the ratio of memory cost to computing time benefit is still worthwhile.

In fact, TurboStencil can solve the stencil computations of general scale on current commercial servers or workstations. Most of these computing devices are equipped with a large capacity of memory, which is sufficient to meet the memory requirement. For example, the Intel Xeon series servers generally have 128 GB of memory and can be expanded to 6TB of memory, which allows TurboStencil to perform with at least 13000×13000 of 2D or $350 \times 350 \times 350$ of 3D double-precision floating-point data.

In addition, we believe that the proposed algorithm is suitable for the computer systems in near future. The emerging memory pooling or disaggregation technology can achieve efficient resource sharing and high-speed communication between the CPU memory and other memory space. Recently released

Table 1
Experimental setup.

Hardware/software	Parameters
Server	Intel(R) Xeon(R) Gold 6248
# of cores	40 cores/socket, 2 sockets
Cache sizes	32KB L1, 1MB L2, 27MB L3(shared)
Memory	188GB
Compiler	icc version 2021.4.0
Compiler flags	-qopenmp -xhost -ansi-alias -ipo -qopt-prefetch=3 -AVX512 -lm -qmkl
Parallelization	OpenMP 201511

Table 2
Benchmarks.

Benchmarks	Stencil points	Typical applications
heat1d	3	transient heat conduction,
jacobi1d	3	fluid dynamics, elastic
heat2d	5	wave simulation for
seidel2d	9	geophysics, numerical
heat3d	7	meteorological simulation,
3d27pt	27	image processing, etc.

Compute Express Link™ (CXL™) 3.0 [27] of the open industry standard interconnect has achieved high-bandwidth, low-latency connectivity, and memory coherency between host processor and attached CXL devices such as accelerators and memory buffers. This enables users to simply focus on the computation rather than the memory requirements. Such advances offer an opportunity for the proposed algorithm to use a unified, expanded memory space and distribute the naïve loop iterative computation to accelerators with trivial cost of data movements.

5. Experiments

We conduct a series of experiments to evaluate the computational performance and numerical accuracy of the TurboStencil algorithms. The experimental setup is list in Table 1.

Baseline method

To the best of our knowledge, there are three convolution-based methods [15–17] for stencil computations, which are similar to ours. Among them, two methods [15,17] employ traditional convolution to compute stencil results, and thus have a high computational complexity of $O(NT)$. And the fast stencil algorithm [16], the state-of-the-art convolution-based method, applies FFT to accelerate the implementation of convolution for stencil computations. But it uses the naïve loop method to compute the results of *affected points* with aperiodic boundary condition. It reduce the computational complexity from $O(NT)$ of naïve loop method to $O(dTN^{1-1/d}\log(dTN^{1-1/d})\log T + N\log N)$. Besides, the fast stencil algorithm also outperforms the state-of-the-art loop tiling-based stencil compiler PLuTo [28,29] by 1.3 × to 8.5 × times for aperiodic stencil problems [16]. Therefore, we choose the fast stencil algorithm as the baseline method for performance evaluation. Both the fast stencil algorithm and our TurboStencil algorithm use the FFT implementation from the Intel Math Kernel Library (Intel MKL) [30].

Benchmarks

We selected 6 benchmarks across 1, 2, and 3 dimensions from [28,29], listed in Table 2. They are stencil computations with aperiodic boundary conditions and widely used in various numerical computing domains. All benchmarks are symmetrical and can be directly tested with the algorithms for symmetric stencil. And the asymmetrical stencil benchmarks are not very common. We manually changed some values of the stencils of benchmarks to make them asymmetrical, and used them to test the algorithms for asymmetric stencil. The differences of values in the stencil pattern do not affect the performance. The input data of benchmarks are double-precision floating-point numbers.

5.1. Performance

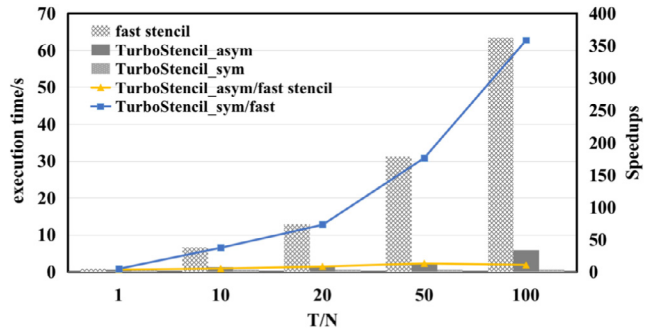
We conducted two different sets of experiments and used the execution time to evaluate the computational performance of our TurboStencil algorithms. We compared the TurboStencil with the *fast stencil* [16] on both symmetric and asymmetric benchmarks. The fast stencil performs the same process on symmetric and asymmetric stencils, so the execution time is the same for both. The *TurboStencil_sym* and *TurboStencil_asym* respectively denote our algorithm for symmetric stencil and asymmetric stencil. Besides, for benchmarks with the same dimension, data size, and time step, tested algorithms have the same execution time for them, which does not vary with different stencil patterns. Therefore, we use 1D stencils, 2D stencils, and 3D stencils respectively to demonstrate the performance of the benchmarks with the same dimension.

In the first set of experiments, we fixed the data size N while varied the time steps T according to different values of $T/N^{1/d}$, d is the number of dimensions. Fig. 11 shows the results. We see that the execution time of *TurboStencil_sym* keeps the same and does not varies with T for the same N. This is because *TurboStencil_sym* only compute once for any time step iterations with a complexity of $O(N\log N)$. However, the execution time of *TurboStencil_asym* grows with T, as *TurboStencil_asym* has to compute T/t times with the complexity of $O((N^{1/d}+2t)^d\log(N^{1/d}+2t)dT/t)$. Despite this, both *TurboStencil_sym* and *TurboStencil_asym* outperform the fast stencil, except the case of $T/N^{1/d} = 1$. For this case with relatively few time steps, the time overhead of memory allocation for padding is much greater than the computation time in our algorithms, but the *fast stencil* does not perform the costly padding. With the increase of T, the performance advantages of our algorithms become more and more significant, while the *fast stencil* suffers the increasingly overhead of iterative computations. And the *fast stencil* has a higher complexity than ours.

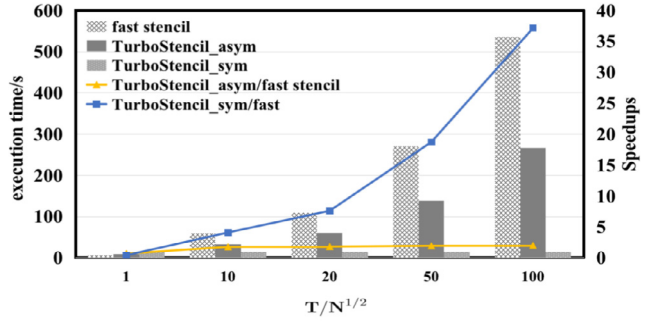
The *TurboStencil_sym* reaches the speedups up to 358.4×, 37.1×, and 14.4× over the *fast stencil* for 1D, 2D, and 3D stencils, respectively. And the corresponding maximum speedups achieved by the *TurboStencil_asym* are 12.9×, 2.0×, and 1.3×, respectively. The more time steps, the better our algorithms perform than the *fast stencil*.

In the second set of experiments, we varied both N and T while fixed the values of $T/N^{1/d} = 100$. Fig. 12 shows the results. For all cases, the *TurboStencil_sym* consistently outperforms the other two algorithms significantly. For the cases of 3D stencils with $N^{1/d} \leq 200$, *TurboStencil_asym* does not perform as well as fast stencil, as the time overhead of the padding method accounts for a relatively large proportion of the overall execution time when data size is small. When $N^{1/d} \geq 250$, *TurboStencil_asym* outperforms *fast stencil*. The *TurboStencil_sym* achieves the speedups up to 777.1×, 43.2×, and 9.4× for 1D, 2D, and 3D stencils, respectively, and the *TurboStencil_asym* achieves the speedups up to 90.4×, 4.6×, and 1.3×. The larger data sizes, the better our algorithms perform. In practical applications [19,31], the data sizes are generally very large, from tens of thousands to even millions, and the time steps are thousands of times of data sizes. This makes our TurboStencil algorithms have promising application prospects.

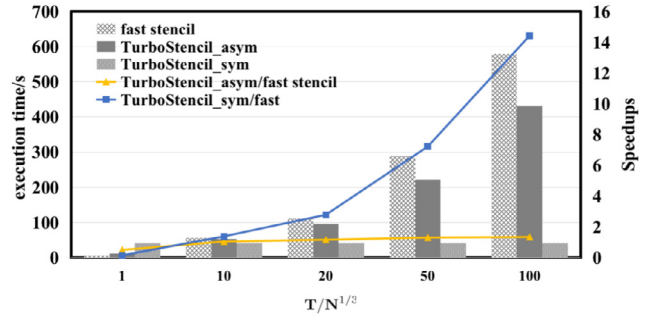
The height of time step band, t, is an important parameter which influences the speed of *TurboStencil_asym*. In the experiments, we manually tuned the optimal t for *TurboStencil_asym*. According to the analysis in Section 4.2, we should choose smaller t for higher dimensional stencils to reduce the time overhead of



(a) 1D stencils with N=500K

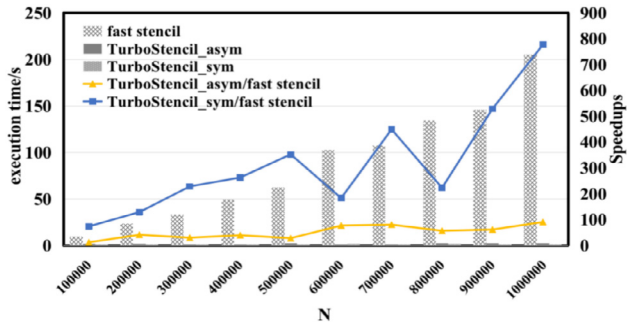


(b) 2D stencils with N=5K×5K

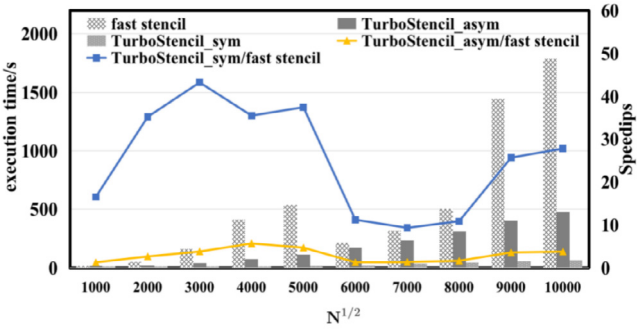


(c) 3D stencils with N=300×300×300

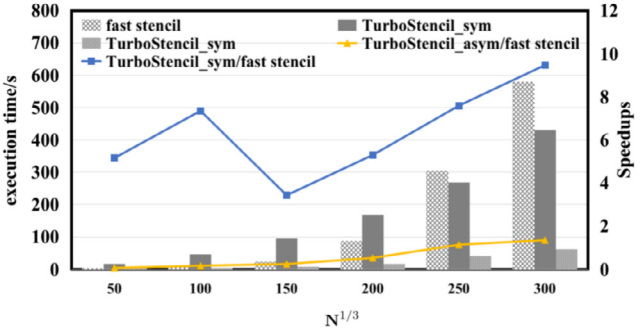
Fig. 11. Performance comparison with state-of-the-art fast stencil algorithm. In the experiments, N was fixed while T was varied.



(a) 1D stencils with T/N=100



(b) 2D stencils with $T/N^{1/2}$=100



(c) 3D stencils with $T/N^{1/3}$ = 100

Fig. 12. Performance comparison with state-of-the-art fast stencil algorithm. In the experiments, both N and T were varied while $T/N^{1/d}$was fixed.

memory allocation (caused by the padding) and FFT computation. For example, when $T/N^{1/d} = 100$ in Fig. 11, we used $t = 250$K for 1D stencils, 2.5 K for 2D stencils, and 50 for 3D stencils.

5.2. Evaluation of general aperiodic boundary condition

To verify the practicality of TurboStencil algorithm applied to the general aperiodic boundary condition, we have also conducted evaluation experiments. Since the fast stencil algorithm is also applicable to this boundary condition, we compared the proposed algorithm with it. Table 3 shows the performance comparison results. The height parameters t of the time step band for TurboStencil are 50000, 2500, and 100 for 1D, 2D, and 3D stencils, respectively. For the general aperiodic boundary condition, TurboStencil needs to performs a small number of naïve loop iterative computations, so there is some degradation in computational performance. However, TurboStencil still outperforms the state-of-the-art fast stencil algorithm by achieving the speedups of 1.74×, 1.34×, and 1.17× for 1D, 2D, and 3D stencils, respectively, in the experiments.

Table 3
Performance comparison with fast stencil for general aperiodic boundary condition.

Benchmarks	Data sizes & Time steps	Execution time (s)		Speedups
		fast stencil	TurboStencil	
1D stencils	100000, 10000000	197.671	113.357	1.74
2D stencils	10000 × 10000, 1000000	538.346	401.566	1.34
3D stencils	500×500×500, 2000	266.724	228.802	1.17

5.3. Numerical accuracy

In this Section, we briefly evaluate the numerical accuracy of our algorithms and the *fast stencil*. We tested maximum absolute error against the output values of naïve loop method in certain time steps. For all benchmarks, input values were set to zeros

Table 4
Numerical accuracy of algorithms with $T = 100$ K.

Benchmarks & Data sizes	Maximum absolute error against naïve loop method		
	fast stencil	*TurboStencil_sym*	*TurboStenicl_asym*
heat1d, 100000	8.15e−10	4.43e−11	2.82e−13
jacobi1d, 100000	1.38e−9	9.02e−10	1.84e−12
heat2d, 1000 × 1000	2.84e−14	1.69e−15	1.56e−11
seidel2d, 1000 × 1000	3.55e−16	6.23e−16	1.58e−11
heat3d, 100 × 100 × 100	<1e−19	3.47e−18	<1e−19
3d27pt, 100 × 100 × 100	<1e−19	2.08e−17	3.20e−18

except the first value was set to one, making the truth value of each point is fallen within [0, 1]. Table 4 shows the results with $T = 100$ K. Because our algorithms and *fast stencil* employ FFT for convolution, they both incur a loss of floating-point accuracy. In the experiment, the maximum absolute errors of these algorithms are close and completely within the acceptance range.

6. Related work

There are three common stencil optimizations: loop tiling, divide-and-conquer methods, and Krylov subspace methods.

Loop tiling. These methods change the iteration order to improve data locality and divide the iteration space into loop tiles to exploit coarse-grained parallelism [9,32,33]. They employ low-level compilation techniques, such as loop transformations, to transform data dependency for developing wave-fronts [34] or concurrent start-up parallelism [7,35,36] for stencil computations. The tile sizes [37,38] also need to be well designed for cache hierarchy efficiency. Based on loop tiling, some source-to-source frameworks, like PLuTo [28], ATF [39] and MSC [1], have been greatly successful in automatically optimizing stencil codes. But these methods cannot reduce the computational complexity of stencil problems.

Divide-and-conquer methods. These methods [10] recursively divide the solution region into sub-regions, and thus achieve cache-oblivious tiling for stencil computations. And they perform redundancy calculations to avoid frequent communication between sub-regions for better parallelism [13]. The Pochoir [14] is a well-known stencil compiler based on this method. It makes hyperspace cuts on grids and yields asymptotically optimal cache efficiency. The divide-and-conquer methods also do not change the total FLOPs of stencil computations.

Krylov subspace methods. Krylov subspace methods are a class of methods that usually used to solve numerical problems. Mathematical techniques are employed to find better approximations of the exact values for stencil problems [11,12]. Such methods generally do not produce exact solutions in finite time, even without floating error, but exhibit a trade-off between computational time and accuracy. The Krylov subspace methods also have to perform costly iterative computations, like the loop tiling and divide-and-conquer methods. Besides, high-level expertise is required for numerical analysis in the design of such methods.

Recently, some **convolution-based computing methods** [15–17] are proposed to fast compute the output of linear stencil computations. These methods apply convolution to the evolution of grid points rather than the iterative computation, and thus significantly reduce the computational time. Januario et al. [15] introduced a convolution-based technique called aggregate stencil-loop iteration, which applies a stencil operator convolved with itself one or more times to speed up stencil computation. Koraei et al. [17] also employs convolution solving stencil problems, but their method works on FPGA. These two methods perform the naïve convolution operations, and thus do not reduce the time complexity. However, convolution provides opportunities to accelerate stencil computations by improving data reuse and FLOPs-to-load ratio. Zafar Ahmad et al. [16] proposed a fast stencil algorithm, which applies FFT to convolution, and thus reduces the time complexity as well as the total FLOPs of stencil with periodic boundary conditions. This state-of-the-art method employs a divide-and-conquer based technique to correct the values of points affected by boundary effect for stencils with aperiodic boundary conditions. And this technique also need to perform naïve loop iterations for some points. On the contrary, we propose a padding method to enable correct convolution for the affected points. Therefore, our method does not explicitly compute any iterations and effectively reduces the time complexity of stencils with aperiodic boundary conditions.

7. Conclusion

In this paper, we present a novel algorithm, named TurboStencil, for fast stencil computation with aperiodic boundary conditions. TurboStencil provides a padding method to eliminate the boundary effect. And our algorithm separately computes the results of points with different stencil patterns by applying FFT to convolution, which enables it to completely avoid explicitly iterative computation. TurboStencil only compute once for symmetric stencil, while also only compute several times for asymmetric stencil. And thus, it is able to effectively reduce the computational complexity. Our experimental results show that TurboStencil significantly outperforms a state-of-the-art fast stencil algorithm for both symmetric and asymmetric stencils, with comparable loss in accuracy. There are still a few classes of stencil computations, including nonlinear stencil and inhomogenous stencil, that are not well addressed. We will further study fast stencil computation algorithms for these problems in the future. Besides, applying TurboStencil to supercomputer systems to solve large-scale applications is also a promising direction of our future work.

CRediT authorship contribution statement

Song Liu: Conceptualization, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing, Funding acquisition. **Xinhe Wan:** Conceptualization, Software, Formal analysis, Methodology, Writing – original draft, Writing – review & editing. **Zengyuan Zhang:** Investigation, Data curation, Visualization. **Bo Zhao:** Investigation, Writing – original draft. **Weiguo Wu:** Supervision, Project administration, Funding acquisition.

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.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 62002279 and No. 61972311) and Natural Science Basic Research Program of Shaanxi (Program No. 2020JQ-077).

References

[1] M. Li, Y. Liu, H. Yang, Y. Hu, Q. Sun, B. Chen, X. You, X. Liu, Z. Luan, D. Qian, Automatic code generation and optimization of large-scale stencil computation on many-core processors, in: 50th International Conference on Parallel Processing, 2021, pp. 1–12.

[2] S. Li, B. Wu, Y. Zhang, X. Wang, J. Li, C. Hu, J. Wang, Y. Feng, N. Nie, Massively scaling the metal microscopic damage simulation on sunway TaihuLight supercomputer, in: Proceedings of the 47th International Conference on Parallel Processing, in: ICPP 2018, Association for Computing Machinery, New York, NY, USA, 2018.

[3] F. Ascione, F. de' Rossi, N. Bianco, G.P. Vanoli, Transient heat transfer through walls and thermal bridges. Numerical modelling: Methodology and validation, in: Proceedings of the Winter Simulation Conference, WSC '12, Winter Simulation Conference, 2012.

[4] B. Lippmeier, G. Keller, Efficient parallel stencil convolution in haskell, in: Proceedings of the 4th ACM Symposium on Haskell, Haskell '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 59–70.

[5] M.B. Nielsen, M. Bojsen-Hansen, K. Stamatelos, R. Bridson, Physics-based combustion simulation, ACM Trans. Graph. 41 (5) (2022) 1–21.

[6] H. Zhou, Y. Liu, J. Wang, Elastic wave modeling with high-order temporal and spatial accuracies by a selectively modified and linearly optimized staggered-grid finite-difference scheme, IEEE Trans. Geosci. Remote Sens. 60 (2021) 1–22.

[7] U. Bondhugula, V. Bandishti, I. Pananilath, Diamond tiling: Tiling techniques to maximize parallelism for stencil computations, IEEE Trans. Parallel Distrib. Syst. 28 (5) (2016) 1285–1298.

[8] V. Bandishti, I. Pananilath, U. Bondhugula, Tiling stencil computations to maximize parallelism, in: Proceedings of the 12th International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2012, pp. 1–11.

[9] I.Z. Reguly, G.R. Mudalige, M.B. Giles, Loop tiling in large-scale stencil codes at run-time with OPS, IEEE Trans. Parallel Distrib. Syst. 29 (4) (2018) 873–886.

[10] T. Gysi, T. Grosser, T. Hoefler, Absinthe: Learning an analytical performance model to fuse and tile stencil codes in one shot, in: 2019 28th International Conference on Parallel Architectures and Compilation Techniques, PACT, IEEE, 2019, pp. 370–382.

[11] O. Andreussi, I. Dabo, N. Marzari, Revised self-consistent continuum solvation in electronic-structure calculations, J. Chem. Phys. 136 (6) (2012) 064102.

[12] V. Fuka, Poisfft–a free parallel fast poisson solver, Appl. Math. Comput. 267 (2015) 356–364.

[13] M. Frigo, V. Strumpen, Cache oblivious stencil computations, in: Proceedings of the 19th Annual International Conference on Supercomputing, 2005, pp. 361–366.

[14] Y. Tang, R.A. Chowdhury, B.C. Kuszmaul, C.-K. Luk, C.E. Leiserson, The pochoir stencil compiler, in: Proceedings of the 23th Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2011, pp. 117–128.

[15] G.C. Januario, B.S. Rosenburg, Y. Park, M. Perrone, J. Moreira, T.C. Carvalho, Speeding up stencil computations with kernel convolution, in: Proceedings of the 28th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD, IEEE, 2016, pp. 76–83.

[16] Z. Ahmad, R. Chowdhury, R. Das, P. Ganapathi, A. Gregory, Y. Zhu, Fast stencil computations using fast Fourier transforms, in: Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures, 2021, pp. 8–21.

[17] M. Koraei, O. Fatemi, M. Jahre, DCMI: A scalable strategy for accelerating iterative stencil loops on FPGAs, ACM Trans. Archit. Code Optim. (TACO) 16 (4) (2019) 1–24.

[18] J. Xiao, S. Li, B. Wu, H. Zhang, K. Li, E. Yao, Y. Zhang, G. Tan, Communication-avoiding for dynamical core of atmospheric general circulation model, in: Proceedings of the 47th International Conference on Parallel Processing, in: ICPP 2018, Association for Computing Machinery, New York, NY, USA, 2018.

[19] H. Fu, J. Liao, N. Ding, X. Duan, L. Gan, Y. Liang, X. Wang, J. Yang, Y. Zheng, W. Liu, L. Wang, G. Yang, Redesigning CAM-SE for peta-scale climate modeling performance and ultra-high resolution on sunway TaihuLight, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17, Association for Computing Machinery, New York, NY, USA, 2017.

[20] A. Zafarullah, Finite difference scheme for a third boundary value problem, J. ACM 16 (4) (1969) 585–591.

[21] M.M. Hasni, Z.A. Majid, N. Senu, Direct 4-point 1-step block method for solving Dirichlet boundary value problem, in: 2015 International Conference on Research and Education in Mathematics, ICREM7, 2015, pp. 76–80.

[22] C.J.W. Tukey, An algorithm for the machine calculation of complex Fourier series, Math. Comp. 19 (90) (1965) 297–301.

[23] A. Lavin, S. Gray, Fast algorithms for convolutional neural networks, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 4013–4021.

[24] N. Brisebarre, M. Joldeş, J.-M. Muller, A.-M. Naneş, J. Picot, Error analysis of some operations involved in the cooley-tukey fast Fourier transform, ACM Trans. Math. Software 46 (2) (2020).

[25] H. Liu, C.-C. Hung, H. Hu, M. Yu, E. Song, An application of circumscribed circle filter in the multi-stencils fast marching method, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 33–38.

[26] E. Raut, J. Anderson, M. Araya-Polo, J. Meng, Porting and evaluation of a distributed task-driven stencil-based application, in: Proceedings of the 12th International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 21–30.

[27] 2023. https://www.computeexpresslink.org/. (Accessed February 2023).

[28] U. Bondhugula, J. Ramanujam, P. Sadayappan, PLuTo: A practical and fully automatic polyhedral program optimization system, in: 2008 29th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2008, pp. 1–15.

[29] Pluto: an automatic parallelizer and locality optimizer for affine loop nests, 2023, http://pluto-compiler.sourceforge.net/. (Accessed February 2023).

[30] Intel, Intel math kernel library, https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html, Inter MKL.

[31] K. Sano, Y. Hatsuda, S. Yamamoto, Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 695–705.

[32] F. Irigoin, R. Triolet, Supernode partitioning, in: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1988, pp. 319–329.

[33] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan, N. Vasilache, Loop transformations: convexity, pruning and optimization, in: Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2011, pp. 549–562.

[34] S. Liu, Y. Cui, N. Zou, W. Zhu, D. Zhang, W. Wu, Revisiting the parallel strategy for DOACROSS loops, J. Comput. Sci. Tech. 34 (2019) 456–475.

[35] T. Grosser, A. Cohen, P.H. Kelly, J. Ramanujam, P. Sadayappan, S. Verdoolaege, Split tiling for GPUs: automatic parallelization using trapezoidal tiles, in: Proceedings of the 6th Workshop on General Purpose Processor using Graphics Processing Units, 2013, pp. 24–31.

[36] S. Shrestha, G.R. Gao, J. Manzano, A. Marquez, J. Feo, Locality aware concurrent start for stencil applications, in: 2015 IEEE/ACM International Symposium on Code Generation and Optimization, CGO, 2015, pp. 157–166.

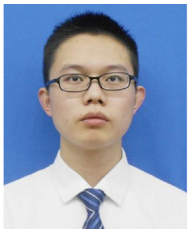
[37] S. Liu, Y. Cui, Q. Jiang, Q. Wang, W. Wu, An efficient tile size selection model based on machine learning, J. Parallel Distrib. Comput. 121 (2018) 27–41.

[38] I.J. Bertolacci, C. Olschanowsky, B. Harshbarger, B.L. Chamberlain, D.G. Wonnacott, M.M. Strout, Parameterized diamond tiling for stencil computations with chapel parallel iterators, in: Proceedings of the 29th ACM on International Conference on Supercomputing, ICS '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 197–206.

[39] A. Rasch, R. Schulze, M. Steuwer, S. Gorlatch, Efficient auto-tuning of parallel programs with interdependent tuning parameters via auto-tuning framework (ATF), ACM Trans. Archit. Code Optim. (TACO) 18 (1) (2021) 1–26.

Song Liu received the B.S. degree in computer science and technology from the Northwestern Polytechnical University, China, in 2009. And he received Ph.D. degree in computer science and technology from the Xi'an Jiaotong University, China, in 2018. He is currently an Assistant Professor at the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China. His research interests include high performance computing and code optimization.

Xinhe Wan received the B.S. degree from the Xi'an Jiaotong University, China, in 2021. He is concurrently working toward the M.S. degree in computer technology at Xi'an Jiaotong University. His research interests include stencil computation and parallel computing.



Zengyuan Zhang received the B.S. degree in computer science and technology from the Xi'an Jiaotong University, China, in 2020. He is concurrently working toward the M.S. degree in computer technology at Xi'an Jiaotong University. His research interests include compiler optimization and parallel computing.

Bo Zhao received the Ph.D. degree from Humboldt-Universität zu Berlin, Germany, in 2022. He is currently an Assistant Professor at Queen Mary University of London and an Honorary Research Fellow at Imperial College London, UK. His research focuses on data-intensive computing systems, including scalable machine learning systems, distributed data management systems, and code optimization techniques.

Weiguo Wu received the B.S., M.S. and Ph.D. degrees in computer science from the Xi'an Jiaotong University, China, in 1986, 1993 and 2006. He is currently a Professor at the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China. His research interests include high performance computer architecture, storage system, cloud computing, and embedded system.