A Fast Algorithm for Aperiodic Linear Stencil Computation using Fast Fourier Transforms

ZAFAR AHMAD and REZAUL CHOWDHURY, Stony Brook University, USA
RATHISH DAS, University of Houston, USA
PRAMOD GANAPATHI, AARON GREGORY, and YIMIN ZHU, Stony Brook University, USA

Stencil computations are widely used to simulate the change of state of physical systems across a multi-dimensional grid over multiple timesteps. The state-of-the-art techniques in this area fall into three groups: cache-aware tiled looping algorithms, cache-oblivious divide-and-conquer trapezoidal algorithms, and Krylov subspace methods.

In this article, we present two efficient parallel algorithms for performing linear stencil computations. Current direct solvers in this domain are computationally inefficient, and Krylov methods require manual labor and mathematical training. We solve these problems for linear stencils by using discrete Fourier transforms preconditioning on a Krylov method to achieve a direct solver that is both fast and general. Indeed, while all currently available algorithms for solving general linear stencils perform $\Theta(NT)$ work, where N is the size of the spatial grid and T is the number of timesteps, our algorithms perform $o(NT)$ work.

To the best of our knowledge, we give the first algorithms that use fast Fourier transforms to compute final grid data by evolving the initial data for many timesteps at once. Our algorithms handle both periodic and aperiodic boundary conditions and achieve polynomially better performance bounds (i.e., computational complexity and parallel runtime) than all other existing solutions.

Initial experimental results show that implementations of our algorithms that evolve grids of roughly 10^7 cells for around 10^5 timesteps run orders of magnitude faster than state-of-the-art implementations for periodic stencil problems, and 1.3× to 8.5× faster for aperiodic stencil problems.

Code Repository: `https://github.com/TEAlab/FFTStencils`

CCS Concepts: • **Mathematics of computing** → **Solvers**; • **Computing methodologies** → **Parallel algorithms**; **Shared memory algorithms**; **Modeling and simulation**; **Linear algebra algorithms;**

Additional Key Words and Phrases: Stencil computation, fast fourier transform, divide-and-conquer, parallel stencil solver

Author list is presented in alphabetical order of last names.

This research was supported by NSF grants CNS-1553510, CCF-1725543, CSR-1763680, CCF-1716252, and CNS-1938709, as well as by the Canada Research Chairs Programme and NSERC Discovery Grants. Part of this work used the Extreme Science and Engineering Discovery Environment (XSEDE) (XSE; Towns et al. 2014), which is supported by NSF grant ACI-1548562. Authors used the XSEDE resources available through startup allocation grant TG-ASC190066.

Authors' addresses: Z. Ahmad, R. Chowdhury, P. Ganapathi, A. Gregory, and Y. Zhu, Stony Brook University, Stony Brook, USA; emails: {zafahmad, rezaul, pramod.ganapathi, afgregory, yimzhu}@cs.stonybrook.edu; R. Das, University of Houston, Houston, USA; email: rathish@central.uh.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

2329-4949/2023/12-ART22 $15.00
https://doi.org/10.1145/3606338

ACM Reference format:
Zafar Ahmad, Rezaul Chowdhury, Rathish Das, Pramod Ganapathi, Aaron Gregory, and Yimin Zhu. 2023. A Fast Algorithm for Aperiodic Linear Stencil Computation using Fast Fourier Transforms. *ACM Trans. Parallel Comput.* 10, 4, Article 22 (December 2023), 34 pages.
https://doi.org/10.1145/3606338

1 INTRODUCTION

A *stencil* is a pattern used to compute the value of a cell in a spatial grid at some timestep from the values of nearby cells at previous timesteps. A *stencil computation* [49, 130] applies a given stencil to the cells in a spatial grid for some set number of timesteps. Stencil computations have applications in a variety of fields, including fluid dynamics [21, 45, 68], electromagnetics [10, 82, 128, 140], mechanical engineering [111, 113, 127], meteorology [11, 75, 114, 115], cellular automata [94, 102, 122, 123], and image processing [107, 117, 142, 146, 147]. In particular, they are widely used for simulating the change of state of physical systems over time [14, 105, 134, 143].

Due to the importance of stencil computations in scientific computing [41, 59, 108], various methods have been devised to improve their runtime performance on different machine architectures [37, 42, 76, 101, 120]. All currently available stencil compilers [25, 37, 65, 67, 92, 109, 130] that can accept arbitrary *linear*[1] stencils perform $\Theta(NT)$ work,[2] where N is the number of cells in the spatial grid and T is the number of timesteps.

In this article, we present the first $o(NT)$-work stencil computation algorithms that support general linear stencils and arbitrary boundary conditions. Our algorithms have polynomially lower work than all other known options of equivalent or greater generality.

Problem Specification. Consider a stencil computation to be performed over T timesteps on a d-dimensional spatial grid of N cells with initial data $a_0[\cdots]$. Cell data at subsequent timesteps are defined via application of the linear stencil S across the grid, formalized as $a_{t+1} = Sa_t$, where a_t is the spatial grid data at timestep t. The stencil must define the value of a grid cell in terms of a fixed size neighborhood containing cells from only the prior timestep.[3] We will not be able to apply the stencil to some cells near the boundaries of the grid; the values of these cells are instead defined via *boundary conditions*. Our goal is to compute the final grid data a_T by evolving the initial data a_0 for T timesteps.

There are two types of boundary conditions we can use: *periodic* and *aperiodic*. If the boundary conditions are periodic, then it means that every dimension of the spatial grid wraps around onto itself, so the entire grid forms a torus. In this case modular arithmetic is used for all calculations involving spatial indices, and the stencil alone can be used to update all cells. However, if the boundary conditions are aperiodic, then the cells at the boundary of the grid have to be computed via some method other than straightforward application of the stencil. In this article, we consider both types of boundary conditions.

Existing Algorithms. There are a handful of algorithms commonly used for carrying out general stencil computations and many more designed for solving problems with specific boundary conditions and stencils. Here we will give an overview of algorithms that can be used for computations with arbitrary boundary conditions and linear stencils. It is worth noting that almost none of the following algorithms require stencil linearity.[4]

[1] A linear stencil is one that uses exclusively linear combinations of grid values from prior timesteps.
[2] Let T_p denote a program's runtime on a p-processor machine. Then T_1 and T_∞ are called *work* and *span*, respectively.
[3] We will later extend the definition of stencils to allow for dependence on multiple prior timesteps.
[4] Although some stencil compilers may not be able to apply their low-level optimizations to nonlinear stencils.

```
STENCIL-LOOP($a_0, S, N, T$)
(1) for $t \leftarrow 1$ to $T$ do
(2)     parallel for $i \leftarrow 0$ to $N-1$ do
(3)         compute $a_t[i]$ by applying either the stencil $S$ or the boundary conditions as appropriate
(4) return $a_T[0, \ldots, N-1]$                              ▷ final spatial grid data
```

Fig. 1. Looping code for one-dimensional stencil computations.

1. [*Looping Algorithms.*] It is a simple matter to implement stencil computations using nested loops, as shown in Figure 1. However, such computations suffer from poor data locality and hence are inefficient both in theory and in practice. As can be seen from the pseudocode listing, looping codes require $\Theta(NT)$ work to evolve a grid of N cells forward T timesteps, assuming that the stencil only uses values from a constant size neighborhood.

2. [*Tiled Looping Algorithms.*] Adopting a tiling [12, 24] of the spatial grid is a common way to improve the data locality [23, 150–153] of looping algorithms. The tiles' size and shape have a strong influence on the runtime of the algorithm, and generally the best performance is attained when each tile fits into the cache. Most modern multicore machines have a hierarchy of caches; to make better use of the cache hierarchy, loop nests may need to be tiled at multiple levels [42, 77].

The most common framework that can be used to derive tiled looping implementations is the *polyhedral model*, which uses a set of hyperplanes to partition the grid being solved for. The polyhedral model is extensively used in several code generators [2–4, 141, 155].

3. [*Recursive Divide-and-Conquer Algorithms.*] Instead of using a predetermined tiling of the spatial grid, these algorithms recursively break the region to be solved for into multiple smaller subregions. The *trapezoidal decomposition algorithm* [50, 51, 131] is the most well-known divide-and-conquer stencil algorithm. Its recursive approach to tiling allows it to be not only both cache oblivious [48] and cache adaptive [16, 17] but also to achieve asymptotic cache performance matching that of an optimally tiled stencil code across all levels of the memory hierarchy in a multicore machine.

4. [*Krylov Subspace Methods.*] Krylov methods compose a diverse set of mathematical techniques that are extensively used in numerical analysis to find successively better approximations of the exact solution to a stencil problem. Such methods are often used to solve problems for which there is no known direct (tiled-loop or divide-and-conquer) solution technique. **Discrete Fourier transforms (DFTs)** are frequently used in the analysis [27] and implementations [7, 62, 74] of these methods. Krylov methods that use DFTs in their implementations are very restricted in their applicability, usually applying only to stencils from specific PDEs that benefit from spectral analysis.

There are several limitations of Krylov methods as a whole: (i) their initial design requires nontrivial manual convergence analysis [29, 87, 103], (ii) they are mostly applicable only to very small classes of problems [8, 56, 74], (iii) they generally do not produce exact solutions in finite time but exhibit a tradeoff between runtime and accuracy. Improving this tradeoff by finding near-optimal preconditioners [18, 19, 135] is a hard problem [33, 81] in the general case.

These common limitations should not be confused for rules, however: Because of their diversity, Krylov methods can take on a variety of useful properties when specially designed. For example, when an optimal preconditioner is selected they can find the exact solution in a finite number of iterations. The algorithms we present in this article will be partially based on an optimally preconditioned Krylov method that is applicable to a rather large class of stencil problems.

Our Fast Fourier Transform–based Algorithms. The computation $S^T a_0 = a_T$ (where S^T denotes that the stencil S is applied T times) evolves the initial grid data a_0 for T timesteps to produce the final data a_T. As will be seen in Section 4, any method of computing $a_{t+1} = Sa_t$ is mathematically equivalent to a product where S is viewed as a matrix and a_t as a vector. All existing algorithms that find a_T exactly do so by direct computation of $a_T = S(S(S(\cdots a_0)))$, where S is applied for a total of T timesteps, incurring $\Theta(NT)$ work in the process. The looping, tiled, and recursive algorithms we have described differ only in how they break up this series of matrix-vector products. We will instead evaluate this product by diagonalization and repeated squaring of the stencil matrix S.

In this article, we present two **fast Fourier transform– (FFT)** based stencil algorithms: StencilFFT-P for problems with periodic boundary conditions and StencilFFT-A for those with aperiodic boundary conditions. Our algorithms are applicable to arbitrary uniform linear stencils across vector-valued fields.

1. [*Periodic Stencil Algorithm.*] Let the DFT matrix be written $\mathcal{F}[i,j] = \omega_N^{-ij}/\sqrt{N}$, where $\omega_N = e^{2\pi\sqrt{-1}/N}$, and let $\mathcal{F}^{-1}$ be the inverse DFT matrix. We use fast Fourier transforms to compute a_T as follows:

$$\boxed{a_T = \mathcal{F}^{-1}(\mathcal{F}S\mathcal{F}^{-1})^T\mathcal{F}a_0},$$

where $\mathcal{F}S\mathcal{F}^{-1}$ is a diagonal matrix and T = #timesteps (not a transposition).

To the best of our knowledge, this is the first time that FFT is being applied directly to the problem of computing integral powers of the circulant [61] stencil matrix S that appears in linear stencil computations, even though there is a strong history of using FFT to improve the efficiency of matrix computations [43, 58, 136].

2. [*Aperiodic Stencil Algorithm.*] When given aperiodic boundary conditions, we use a recursive divide-and-conquer strategy to solve for the boundary of the spatial grid; StencilFFT-P is used as a subroutine to compute cells whose values are independent of the boundary. This method allows us to compute every timestep of the boundary in serial and yet to skip over computing most timesteps of cells near the middle of the grid.

Before we analyze the complexities of our algorithms briefly described above, we give the performance metrics that will be used.

Performance Metrics. We use the *work-span* model [39] to analyze the performance of dynamic multithreaded parallel programs. *Work* $T_1(n)$ of an algorithm, where n is the input parameter, denotes the total number of serial computations. *Span* $T_\infty(n)$ of an algorithm, also called *critical-path length* or *depth*, denotes the maximum number of operations performed on any single processor when the algorithm is run on a machine with an unbounded number of processors. Our analysis of span is performed according to the *binary-forking* model [22], in which spawning n threads required $\Theta(\log n)$ span. This model is stricter than PRAM, so all bounds we give hold in the PRAM model as well. *Parallel running time* $T_p(n)$ of an algorithm when run on p processors under a greedy scheduler is given by $T_p(n) = O(T_1(n)/p + T_\infty(n))$. *Parallelism* of an algorithm is the average amount of work performed in each step of its critical path and is computed as $T_1(n)/T_\infty(n)$.

Performance Analysis of Our Algorithms. The performance of our periodic and aperiodic stencil algorithms are summarized in Table 1. We see that (i) both work and span of StencilFFT-P have only logarithmic dependence on T compared with the linear dependence on T in the existing algorithms. (ii) For a d-dimensional problem, StencilFFT-A has work quasilinearly dependent on $(TN^{1-1/d} + N)$, whereas all existing algorithms for general linear stencils perform $\Theta(NT)$ work—a polynomially greater amount. This asymptotic improvement makes possible stencil computations over much larger spacetime grids.

Although we do not show explicit analysis of cache complexity in this article, it is worth noting that our algorithms are cache oblivious [48] and cache adaptive [16, 17].

Table 1. Complexity Bounds for Stencil Algorithms, where N = Spatial Grid Size, T = #Timesteps, and M = Cache Size

Algorithm	Work (T_1)	Span (T_∞)	Result
Existing Algorithms			
Nested Loop	$\Theta(NT)$	$\Theta(T \log N)$	
Tiled Loop	$\Theta(NT)$	$\Theta\left(T \log M + \frac{T}{M^{1/d}} \log \frac{N}{M}\right)$	[12, 24]
Divide-and-Conquer	$\Theta(NT)$	$\Theta\left(T (N^{1/d})^{\log(d+2)-1}\right)$	[131]
Our Algorithms			
StencilFFT-P	$\Theta(N \log(NT))$	$\Theta(\log T + \log N \log\log N)$	Theorem 4.1
StencilFFT-A	$\Theta\left(\begin{array}{c} TN^{1-1/d} \log\left(TN^{1-1/d}\right) \log T \\ +N \log N \end{array}\right)$	$\begin{cases} \Theta(T) & \text{if } d = 1 \\ \Theta(T \log N) & \text{if } d \geq 2 \end{cases}$	Theorem 5.1

It is important to note that the bounds given for our algorithms are for computations with $\Theta(1)$-size stencils on $d = \Theta(1)$ dimensional hypercubic grids and that we have simplified the span of StencilFFT-A by assuming $T = \Omega(\log N \log\log N)$. The nested loop, tiled loop, and D&C algorithms work for both periodic and aperiodic boundary conditions. The span for the tiled loop algorithm is $\Omega(T \log\log N)$.

Our Contributions. Our key contributions are as follows:
1. [*Theory.*] We present the first algorithms for general linear stencil computations (for both periodic and aperiodic boundary conditions) with $o(NT)$ work and low span, achieving polynomial speedups over the best existing stencil algorithms.
2. [*Practice.*] We experimentally analyze the numerical accuracy and runtime of our algorithms as compared to PLuTo [3] code. Implementations of our algorithms for on the order of 10^7 grid cells and 10^5 timesteps suffer no more loss in accuracy from floating point arithmetic than PluTo code yet run orders of magnitude faster than the best existing implementations of state-of-the-art algorithms for periodic stencil problems and 1.3× to 8.5× faster for aperiodic stencil problems. This is shown in Table 2. Our code is publicly available at
https://github.com/TEAlab/FFTStencils.

2 RELATED WORK AND ITS LIMITATIONS

There is substantial literature devoted to the applications and analysis of stencils and DFTs. Here we give a background of the relationship between these two areas and examine some of the limitations that appear in the current approaches to stencil codes.

Discrete Fourier Transforms. DFTs are widely used in numerical analysis, with examples including Von Neumann stability analysis [106, 148] to show validity of numerical schemes, DFT-based preconditioning to optimize Krylov iterations [31, 63, 79], and time-domain analysis to achieve partial solutions of given PDEs [36, 69, 97, 104, 121].

An FFT is an algorithm that quickly computes the DFT of an array. The use of FFTs will be important for our algorithms, as they represent a particularly efficient type of matrix-vector multiplication. Several $O(N \log N)$-work FFT algorithms exist [30, 60, 149], the most famous among which is the Cooley-Tukey algorithm [38].

Theorem 2.1 (Cooley-Tukey Algorithm, [38, 48]). *The generic Cooley-Tukey FFT algorithm computes the DFT of an array of size N in $\Theta(N \log N)$ work, $\Theta(\log N \log\log N)$ span, and $\Theta(N)$ space.*

Stencil Problems. Stencils are often used in numerical analysis as discretizations of PDEs, since many simple PDEs have prohibitively complex analytical solutions [46, 110] but allow good numerical approximations with a proper choice of stencil.

Table 2. Performance Summary of Parallel Stencil Algorithms on a KNL/SKX Node

Benchmark		Stencil	N	T	Parallel runtime in seconds: PLuTo KNL	PLuTo SKX	Our algorithm KNL	Our algorithm SKX	Speedup factor over PLuTo KNL	Speedup factor over PLuTo SKX
Periodic		heat1d	1,600,000	10^6	79	19	0.25	0.03	1754.7	759.6
Periodic		heat2d	$8{,}000 \times 8{,}000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
Periodic		seidel2d	$8{,}000 \times 8{,}000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
Periodic		jacobi2d	$8{,}000 \times 8{,}000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
Periodic		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
Periodic		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	1,600,000	10^6	50	35	5.85	6.69	8.5	5.2
Aperiodic	Experiment 1	heat2d	$8{,}000 \times 8{,}000$	10^5	333	530	143.25	151.37	2.3	3.5
Aperiodic	Experiment 1	seidel2d	$8{,}000 \times 8{,}000$	10^5	345	601	145.42	132.97	2.4	4.5
Aperiodic	Experiment 1	jacobi2d	$8{,}000 \times 8{,}000$	10^5	567	456	249.04	273.46	2.3	1.7
Aperiodic	Experiment 1	heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
Aperiodic	Experiment 1	19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
Aperiodic	Experiment 2	heat1d	1,600,000	N	32	23	5.63	6.87	5.7	3.3
Aperiodic	Experiment 2	heat2d	$8{,}000 \times 8{,}000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
Aperiodic	Experiment 2	seidel2d	$8{,}000 \times 8{,}000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
Aperiodic	Experiment 2	jacobi2d	$8{,}000 \times 8{,}000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
Aperiodic	Experiment 2	heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
Aperiodic	Experiment 2	19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

There are two major types of methods related to stencils: those for deriving numerical schemes and those for evolving grid data via a given stencil. A *discretization method* is a way of converting a PDE, which deals with quantities defined over a continuum, into a stencil, which relates quantities defined over discrete sets of variables. A *stencil solver* is an algorithm that takes stencils, boundary conditions, and initial data as input and performs stencil computations to output final data. This article presents a pair of stencil solvers for linear stencils that support, respectively, periodic and aperiodic boundary conditions.

Stencil Solvers. Numerical results from stencils are obtained through two primary paths: *direct solvers* and *Krylov methods* [70, 118].

Direct solvers are those that find the solution to a stencil problem in a finite number of steps. They often involve feeding the stencil into a stencil compiler such as PLuTo [25], Pochoir [130], or Devito [91], which will output optimized code to compute the action of the stencil across some prespecified grid of initial data for multiple timesteps. Cutting-edge stencil code generators feature many improvements over simple looping algorithms, including better cache efficiency [49, 85], parallelism [83], and low-level compiler optimizations. These systems all perform the same set of updates on the stencil grid, although they vary in the order that these updates are performed. *In general, they make no use of FFTs.*

Krylov subspace methods produce successively better approximations of the exact solution to a given stencil problem. Krylov solvers are often used to solve problems for which there is no known direct solution technique. It is common for DFTs to be used either in the analysis [27] or implementations [7, 62, 74] of Krylov subspace methods, as Fourier analysis is useful for proving scheme stability [32, 156] and convergence rates [57], and DFT matrices are good preconditioners

[44] for a large class of matrix equations [26]. In some instances choosing the DFT matrix as a Krylov preconditioner can even convert an approximate solver into a direct one [54, 69]. Krylov subspace methods generally do not produce exact solutions in finite time but exhibit a *tradeoff between runtime and accuracy.*

Limitations of Current Methods for Stencil Problems.

(1) ***Manual Analysis.*** Krylov subspace methods are often accompanied by a mathematically nontrivial convergence analysis [29, 87, 103]; this requires human labor for every new method developed. Since this analysis is not automated [53, 64, 99], the quantity of time it takes varies widely from case to case. Also, the requirement of mathematical rigour strongly discourages the development of unnecessarily general Krylov methods, thus those in the literature usually only apply to specific stencils [8, 9, 56, 93] to simplify analysis.

(2) ***Specialization.*** The methods published in most of the existing literature on computation with numerical schemes are only applicable to very small classes of problems [8, 56, 74, 121]. DFT preconditioning for Krylov iterations has been used for specific stencils before [52, 72, 124]. However, these techniques have not been generalized to work for higher dimensional grids with general linear stencils.

(3) ***Inexact Solution.*** Krylov methods often cannot produce exact solutions, even in the absence of floating-point rounding errors. Using them optimally and reliably thus requires expertise in numerical analysis [70, 137], as well as for substantial effort to be put into uncertainty quantification [86, 89].

(4) ***Nonoptimal Computational Complexity.*** All currently available code compilers [25, 37, 65, 67, 92, 109, 130] generate code that has linear work complexity in the number of grid cells and number of timesteps to compute, no matter what stencil they are given. Improving this bound has not been addressed in the literature, even when only considering linear stencils.

We show in this article that when dealing with linear stencils it is possible to produce code that has significantly better asymptotic performance.

(5) ***No Support for Implicit Stencils.*** Direct solvers for stencil problems usually do not support implicit stencils; stencil compilers in particular are weak in this respect. Pochoir, PLuTo, and Devito cannot be used for directly evolving data via an implicit stencil. This is a significant limitation, as several important stencils are implicit [6, 40, 80, 100, 115, 138].

Significance of This Article. Current direct solvers for general linear stencil computations are inefficient, and Krylov methods require manual labor and mathematical training. We solve these problems for linear stencils by using DFT preconditioning on a Krylov method to achieve a direct solver that is both fast and general.

3 APPLICABILITY OF OUR ALGORITHMS

In this section, we describe the classes of stencil problems on which our FFT-based stencil algorithms do or do not apply.

Supported Stencil Types. Our algorithms are most directly applicable to *homogeneous linear stencils across vector-valued fields.* A homogeneous stencil is one that does not vary across the entire spatial grid, and by vector-valued fields we mean we allow each cell value across the spatial grid to be treated as a vector.

All homogeneous linear PDEs can be discretized into supported stencils by using a finite difference approximation [71]. Thus all numerical results for these linear PDEs that were previously reached via analytically motivated numerical schemes, including those set in the Fourier domain [95, 97], can easily be reached computationally by our algorithms.

It is noteworthy that we support both explicit and implicit linear stencils and also that vector-valued fields can be used to enable stencils that are dependent on more than one previous timestep. In addition, vector-valued fields allow us to support certain types of inhomogeneity, as mentioned at the end of this section.

Linear stencils are quite common in computational numerics. In fact, the majority of stencils currently used to benchmark [3, 23, 25, 112] stencil compilers are linear.

There may come times when we wish to apply a stencil to some spatial grid until a *stopping condition* is met, such as the field data converging to a steady-state solution or numerically destabilizing. If we assume the condition to hold for all timesteps after some cutoff T, then we can use our FFT-based algorithm with a $\Theta(\log T)$ factor overhead to find that cutoff. We first do an exponential search using our FFT-based algorithm, i.e., running it for 2^i timesteps in the ith attempt ($i \geq 0$), checking for the stopping condition after every attempt $i > 0$, and stopping as soon as the condition is met. If we stop after the jth attempt, then the cutoff $T \in (2^{j-1}, 2^j]$ and we perform a binary search within that range to find T. This requires running our FFT-based algorithm $\Theta(\log T)$ times to reach the earliest stopping point that satisfies specific conditions potentially orders of magnitude faster than would be possible with looping code.

Unsupported Stencil Types. Our algorithms are not applicable to *nonlinear stencils*. This is because introducing nonlinearity of any sort invalidates our technique of using DFTs to simplify the action of the stencil. Common examples of nonlinearity in stencils include conditionals, i.e., max/min/if-else, and quadratic dependence on cell values. Most discretizations of nonlinear PDEs pass the nonlinearity on to the stencil, so in general our algorithms cannot be used for stencils from nonlinear PDEs.

Our algorithms cannot be applied to *inhomogeneous stencils* either. There are two ways that a stencil can break homogeneity. The first is spatially, by having the stencil itself be dependent on local field data, such as is used in slope limiter and flux limiter methods [55, 88, 126] and in mixed media problems [34, 119, 132, 133, 144, 154]. The second way to break homogeneity is temporally, i.e., using a stencil that is dependent on time [66, 84], as would arise from the presence of a forcing term in the original PDE being discretized.

However, we note that there are some special types of inhomogeneity our algorithms can handle, such as those arising from forcing terms that are low-order polynomials in time. These are handled by discretizing homogeneous systems of PDEs to mimic the behaviour of a single inhomogenous PDE.

4 PERIODIC STENCIL ALGORITHM

In this section, we present StencilFFT-P, an efficient parallel algorithm for performing stencil computations with periodic boundary conditions using FFT. We begin by considering explicit linear stencils on one-dimensional spatial grids, after which we give simple extensions to high-dimensional grids, implicit stencils, and grids where cells are vector valued.

Mathematical Formulation. Suppose we have a spatial grid of data that evolves in time according to some fixed stencil: Cells in the grid at time t are updated as a function of some local neighborhood of cell values at recent times before t. For simplicity's sake, we will assume that the spatial grid is *one dimensional* until Section 4.2.

In this section, we will exclusively consider linear stencils. A *linear stencil* S defines future array values $a_{t+1}[0, \ldots, N-1]$ as a linear function of current array values $a_t[0, \ldots, N-1]$. We will later allow array values to be higher dimensional, but for now these constraints are enough on their own to make a significant statement about stencils.

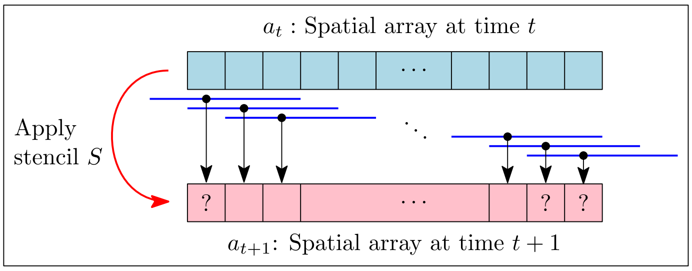
a_t : Spatial array at time t
Apply stencil S
?
?
?
a_{t+1}: Spatial array at time $t+1$

Fig. 2. Updating data with a stencil that uses a neighborhood extending one cell to the left and two cells to the right. Cells marked with question marks do not have values defined by the stencil, as the neighborhood required to update them lies partially outside the bounds of the spatial array.

An arbitrary linear mapping from arrays of size N to arrays of size N is, by definition, an $N \times N$ ***matrix***,[5] and therefore the update rule can be written $a_{t+1}[i] = \sum_j S[i,j]a_t[j]$. We will usually omit the indices in formulas like this, writing $a_{t+1} = Sa_t$. As shown in Figure 2, on the surface this update rule may look incomplete—since we require S to use the same neighborhood around each point for updates, how should we proceed when the neighborhood extends beyond the bounds of our spatial array? As filling in these cells is exactly the purpose of boundary conditions, it should come as no surprise that in this section the resolution to this difficulty will come in the form of periodicity.

Periodic boundary conditions consist of the rule that $a_t[i] = a_t[i+N]$, for all i. This allows the spatial grid to be extended arbitrarily far in either direction by wrapping around instead of moving outside of the array bounds; in the presence of periodic boundary conditions the spatial grid is effectively a torus, with no clearly defined boundary.

For a stencil to be uniform across space means that it defines updated cell values only from a set of cells that are selected based on their *relative* location to the cell being updated. For example, if we were to reindex all cells in the spatial array, incrementing them all so the bounds became 1 and N rather than 0 and $N-1$, then this change of index ought to be invisible to the stencil. Furthermore, in the presence of periodic boundary conditions, this reindexing is equivalent to cyclically shifting the field data a_t, since we have $a_t[0] = a_t[N]$.

We mathematically represent the concept of cyclically permuting grid data by introducing the *right shift matrix*[6] [116] X. The array Xa_t is defined to be the result of taking the rightmost element $a_t[0]$ off and appending it to the left side of the array, i.e., its action on arrays is $(Xa_t)[i] = a_t[i-1]$. An equivalent definition is by X's matrix elements $X[i,j] = \delta_{i,j+1}$, where the *Kronecker delta* $\delta_{i,j}$ is defined to be 1 if $i = j$ and 0 otherwise, and the arithmetic is understood to be *modular* with base N in the presence of periodic boundary conditions.

Given that we have periodic boundary conditions and the update rule is fixed across space, the action of our stencils must be invariant under spatial shifts of the grid values. As shown in Figure 3, cyclically permuting a_t and then applying S should give the exact same result as applying S and then cyclically permuting a_{t+1}. In symbols, we have $SX = XS$, which implies that S must be a *circulant matrix* [61], satisfying $S[i,j] = S[i-j,0]$. If we name these elements $S[i-j,0] = s_{i-j}$, then

[5]Here we are using the word matrix in the strictly *mathematical* sense, i.e., the object that is a stencil behaves in all respects identically to the way in which a matrix behaves. This should not be taken to mean that we will store stencils using the *data structure* called a matrix. In fact, we shall show that there are other more efficient ways of storing our stencils.
[6]Under periodic boundary conditions, shifting the array is equivalent to rotating the array.

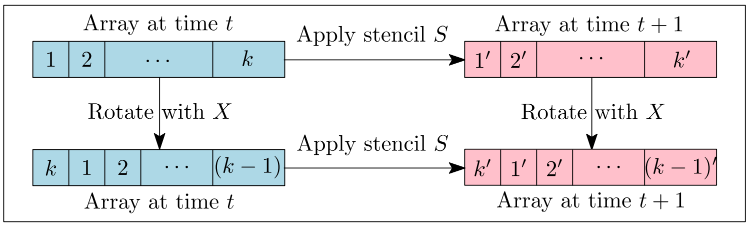

Fig. 3. The stencil S does not care where cells are with respect to the start of the array; it only cares about where they are with respect to one another. The shift matrix X does not affect cells' relative locations, so it does not change the values of the updated cells.

we can write out the full update equation as follows:

$$\underbrace{\begin{bmatrix} a_{t+1}[0] \\ a_{t+1}[1] \\ \vdots \\ a_{t+1}[N-2] \\ a_{t+1}[N-1] \end{bmatrix}}_{a_{t+1}} = \underbrace{\begin{bmatrix} s_0 & s_{N-1} & \cdots & s_2 & s_1 \\ s_1 & s_0 & s_{N-1} & & s_2 \\ \vdots & s_1 & s_0 & \ddots & \vdots \\ s_{N-2} & & \ddots & \ddots & s_{N-1} \\ s_{N-1} & s_{N-2} & \cdots & s_1 & s_0 \end{bmatrix}}_{S} \underbrace{\begin{bmatrix} a_t[0] \\ a_t[1] \\ \vdots \\ a_t[N-2] \\ a_t[N-1] \end{bmatrix}}_{a_t}.$$

An example of this matrix with concrete numbers will now be given. Suppose we have a one-dimensional (1D) periodic spatial grid of four cells, and we want to act on it with a stencil S that computes the new value of a cell according to $a_{t+1}[i] = -2a_t[i-1] + a_t[i] + 3a_t[i+1]$. We can write out the update equation in full as

$$\underbrace{\begin{bmatrix} a_{t+1}[0] \\ a_{t+1}[1] \\ a_{t+1}[2] \\ a_{t+1}[3] \end{bmatrix}}_{a_{t+1}} = \underbrace{\begin{bmatrix} 1 & 3 & 0 & -2 \\ -2 & 1 & 3 & 0 \\ 0 & -2 & 1 & 3 \\ 3 & 0 & -2 & 1 \end{bmatrix}}_{S} \underbrace{\begin{bmatrix} a_t[0] \\ a_t[1] \\ a_t[2] \\ a_t[3] \end{bmatrix}}_{a_t}.$$

For higher-dimension visualizing, the circulant nature of periodic grids can be tricky. Figure 4 shows all the rotations of a circulant spatial grid in 2D.

A useful representation of circulant matrices is found through the right shift matrix. Notice that powers of X have the property $(X^k a_t)[i] = a_t[i-k]$, which means that their matrix elements are given by $(X^k)[i,j] = \delta_{i,j+k}$. Thus powers of X allow us to pick out the individual diagonals that appear in circulant matrices. In fact, any circulant matrix S can be expanded in terms of shift operators as

$$S = \sum_i S[i,0]X^i,$$

because for any vector a, we have

$$\begin{aligned}(Sa)[i] &= \sum_j S[i,j]a[j] = \sum_j S[i-j,0]a[j] \\ &= \sum_j S[j,0]a[i-j] = \sum_j S[j,0](X^j a)[i] \\ &= \left(\left(\sum_j S[j,0]X^j\right) a\right)[i].\end{aligned}$$

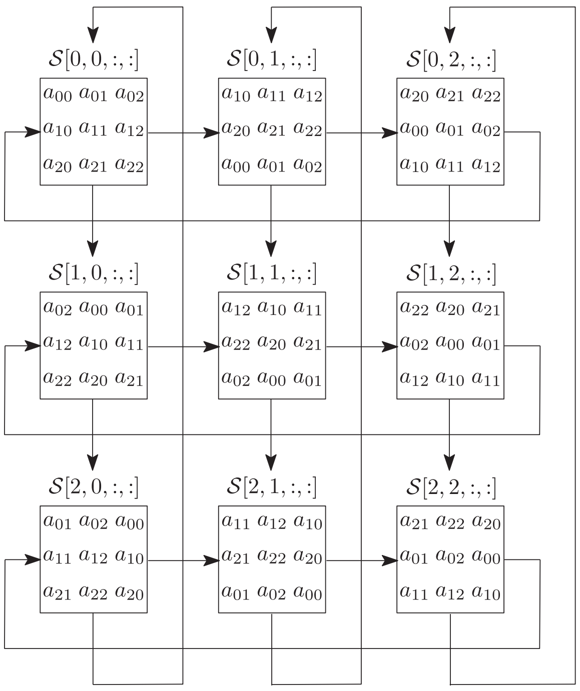
$\mathcal{S}[0,0,:,:]$
$a_{00}\ a_{01}\ a_{02}$ / $a_{10}\ a_{11}\ a_{12}$ / $a_{20}\ a_{21}\ a_{22}$
$\mathcal{S}[0,1,:,:]$
$a_{10}\ a_{11}\ a_{12}$ / $a_{20}\ a_{21}\ a_{22}$ / $a_{00}\ a_{01}\ a_{02}$
$\mathcal{S}[0,2,:,:]$
$a_{20}\ a_{21}\ a_{22}$ / $a_{00}\ a_{01}\ a_{02}$ / $a_{10}\ a_{11}\ a_{12}$
$\mathcal{S}[1,0,:,:]$
$a_{02}\ a_{00}\ a_{01}$ / $a_{12}\ a_{10}\ a_{11}$ / $a_{22}\ a_{20}\ a_{21}$
$\mathcal{S}[1,1,:,:]$
$a_{12}\ a_{10}\ a_{11}$ / $a_{22}\ a_{20}\ a_{21}$ / $a_{02}\ a_{00}\ a_{01}$
$\mathcal{S}[1,2,:,:]$
$a_{22}\ a_{20}\ a_{21}$ / $a_{02}\ a_{00}\ a_{01}$ / $a_{12}\ a_{10}\ a_{11}$
$\mathcal{S}[2,0,:,:]$
$a_{01}\ a_{02}\ a_{00}$ / $a_{11}\ a_{12}\ a_{10}$ / $a_{21}\ a_{22}\ a_{20}$
$\mathcal{S}[2,1,:,:]$
$a_{11}\ a_{12}\ a_{10}$ / $a_{21}\ a_{22}\ a_{20}$ / $a_{01}\ a_{02}\ a_{00}$
$\mathcal{S}[2,2,:,:]$
$a_{21}\ a_{22}\ a_{20}$ / $a_{01}\ a_{02}\ a_{00}$ / $a_{11}\ a_{12}\ a_{10}$

Fig. 4. Visualization of circulant components of a 2D periodic spatial grid.

The above equation shows that circulant matrices can be uniquely specified by a single one of their columns or rows. We will make use of this fact to avoid performing redundant computations: For all of the algorithms presented in this article, we will store *only the first column* of S in memory.

Reformulating the Final Data. We now turn back to the definition of the final data $a_T = S^T a_0$ and the DFT matrix $\mathcal{F}$. Here the exponent T will always denote a matrix power, *not a transposition.* As before, the DFT matrix has elements $\mathcal{F}[i,j] = \omega_N^{-ij}$, where $\omega_N = e^{2\pi\sqrt{-1}/N}$ is a primitive Nth root of unity, and $\mathcal{F}$'s inverse has elements $\mathcal{F}^{-1}[i,j] = \omega_N^{ij}/N$. Since we know that $\mathcal{F}^{-1}\mathcal{F}$ is the identity, it can make no difference mathematically to drop it into our equation for the final data:

$$a_T = \mathcal{F}^{-1}\mathcal{F}S^T\mathcal{F}^{-1}\mathcal{F}a_0.$$

Continuing to insert identities and regrouping, we find that $\mathcal{F}S^T\mathcal{F}^{-1} = \mathcal{F}S\mathcal{F}^{-1}\mathcal{F}S\mathcal{F}^{-1}\cdots\mathcal{F}S\mathcal{F}^{-1} = (\mathcal{F}S\mathcal{F}^{-1})^T$, so we can rewrite our equation as

$$a_T = \mathcal{F}^{-1}(\mathcal{F}S\mathcal{F}^{-1})^T\mathcal{F}a_0.$$

This form of the final data equation points to a remarkably efficient way of computing a_T. We first apply the *convolution theorem* [28], which states that if S is a circulant matrix, then $\mathcal{F}S\mathcal{F}^{-1} = \Lambda$ is diagonal. The equation for final data can now be regrouped with $\mathcal{F}S\mathcal{F}^{-1} = \Lambda$ and $\mathcal{F}a_0 = x$:

$$\boxed{a_T = \mathcal{F}^{-1}\Lambda^T x.} \tag{1}$$

This equation may appear to be more complicated than what we started with, but really all we have done is made a change of basis into the frequency domain and performed all actions of the stencil there. This will now be shown to be computable with only a couple calls to highly efficient FFTs and some repeated squarings of scalars.

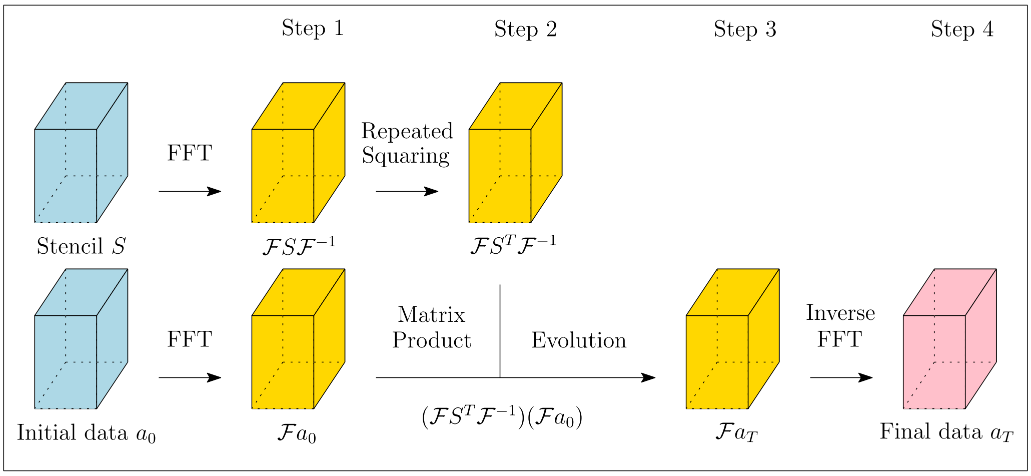

Fig. 5. Block diagram of our FFT-based periodic stencil algorithm, which works for all grid sizes in all dimensions.

4.1 One-dimensional Explicit Stencil Algorithm

Let $a_0[0,\ldots,N-1]$ be the initial 1D spatial grid data to be acted on with the stencil $S = \sum_i S[i,0]X^i$. We will impose periodic boundary conditions; these allow us to benefit from pushing the stencil and initial data into the frequency domain with $\mathcal{F}S\mathcal{F}^{-1} = \Lambda$ and $\mathcal{F}a_0 = x$. Our goal is to compute the final data

$$a_T = S^T a_0 = \mathcal{F}^{-1}\Lambda^T x.$$

The prevalent approach to finding a_T is currently through iterative applications of the stencil to field data, grouping $S^T a_0$ into $S(S(\cdots S(a_0)\cdots))$ and evaluating according to parenthesization. Here we will instead compute a power of the diagonalized stencil $(\mathcal{F}S\mathcal{F}^{-1})^T = \Lambda^T$ by repeated squaring, after which we will apply it to $\mathcal{F}a_0$, giving us $\mathcal{F}S^T a_0$, from which we can recover $S^T a_0$ with an inverse FFT.

It is shown in Section 4.2 that we can write the elements of the diagonal matrix Λ as $\Lambda[i,i] = (\mathcal{F}s)[i]$, where s is the column of S that we are storing in memory, i.e., $s[i] = S[i,0]$. Since $\mathcal{F}$ is the DFT matrix, this means that Λ can be computed with a single FFT.

Blindly using repeated squaring only allows us to compute Λ^T when T is an exact power of 2; arbitrary positive integer powers are computed as follows. Let $\sum_i b_i 2^i$ be the binary representation of T. As we compute successive squares of Λ, i.e., Λ^{2^i}, we will multiply them into a running total only if $b_i = 1$. Thus the final result will be

$$\Lambda^T = \prod_{i:\, b_i=1} \Lambda^{2^i}.$$

Since Λ is diagonal, elements of the large matrix power Λ^T can be computed by taking powers of the original elements, $\Lambda^T[i,i] = \Lambda[i,i]^T$. Evaluating each of these elements in parallel will improve the span of our algorithm.

Wrapping up, we evaluate Equation (1) as follows: We find $x = \mathcal{F}a_0$ by applying FFT to the initial data, $\Lambda^T x$ by elementwise multiplication, and then a_T with an inverse FFT.

We now present the PStencil-1D-FFT algorithm, which efficiently performs stencil computations with periodic boundary conditions by transferring almost all relevant calculations to within the frequency domain. A diagrammatic outline is shown in Figure 5, and the pseudocode is given in Figure 6.

```
STENCILFFT-P($s, a_0, \ell_1, \ldots, \ell_d, T$)
     [Step 1. FFT.] ________ From $S$ to $\Lambda = \mathcal{F}S\mathcal{F}^{-1}$, from $a_0$ to $x = \mathcal{F}a_0$.
(1)  $\Lambda \leftarrow$ MULTI-FFT($s$)
(2)  $x \leftarrow$ MULTI-FFT($a_0$)
     [Step 2. Repeated Squaring.] ________ From $\Lambda$ to $V = \Lambda^T$.
(3)  $V \leftarrow$ Array of size $\ell_1 \times \cdots \times \ell_d$ initialized with all 1s
(4)  $R \leftarrow$ Array of size $\ell_1 \times \cdots \times \ell_d$ initialized with all $T$s
(5)  Squares $\leftarrow \Lambda$                              ▷ We'll store $\Lambda^{2^k}$ in Squares
(6)  parallel for $j_1 \leftarrow 0$ to $\ell_1 - 1$ do
       $\ddots$
(7)    parallel for $j_d \leftarrow 0$ to $\ell_d - 1$ do
(8)      for $k \leftarrow 0$ to $\log T$ do
(9)        if $R[j_1..j_d]$ is odd then                ▷ Picking out binary representation of $T$
(10)         $V[j_1..j_d] \leftarrow V[j_1..j_d] \times$ Squares$[j_1..j_d]$
(11)         $R[j_1..j_d] \leftarrow R[j_1..j_d] - 1$
(12)       $R[j_1..j_d] \leftarrow R[j_1..j_d]/2$
(13)       Squares$[j_1..j_d] \leftarrow$ Squares$[j_1..j_d]^2$
     [Step 3. Convolution.] ________ From $V = \Lambda^T$ and $x$ to $y = \Lambda^T x$.
(14) $y \leftarrow$ Array of size $\ell_1 \times \cdots \times \ell_d$
(15) parallel for $j_1 \leftarrow 0$ to $\ell_1 - 1$ do
         $\ddots$
(16)     parallel for $j_d \leftarrow 0$ to $\ell_d - 1$ do
(17)       $y[j_1..j_d] \leftarrow V[j_1..j_d] \times x[j_1..j_d]$
     [Step 4. Inverse FFT.] ________ From $y = \mathcal{F}a_T$ to $a_T$.
(18) $a_T \leftarrow$ INVERSE-MULTI-FFT($y$)
```

Fig. 6. Arbitrary dimensional periodic stencil algorithm. The MULTI-FFT algorithm takes FFTs over every index of the array passed to it. Lines 6–13 compute $\Lambda[j_1..j_d]^T$, where T need not be an exact power of two.

[**Step 1. FFT.**] We compute (i) $\mathcal{F}S\mathcal{F}^{-1}$ from S and (ii) $\mathcal{F}a_0$ from a_0. Since S is circulant, we know that the FFT of S's first column contains exactly the same information as $\mathcal{F}S\mathcal{F}^{-1}$. Thus for (i) an FFT is applied to the first column of S to get S's eigenvalues; this FFT will be computed for N points, irrespective of how many nonzero coefficients are present in the stencil. Note that only the first column of S is needed here, which is why the rest of S is never constructed or stored in memory. Likewise, for (ii) an FFT is applied to a_0.

[**Step 2. Repeated Squaring.**] We compute $\mathcal{F}S^T\mathcal{F}^{-1}$ from $\mathcal{F}S\mathcal{F}^{-1}$. Since $\mathcal{F}S\mathcal{F}^{-1}$ is diagonal, the individual elements of $\mathcal{F}S^T\mathcal{F}^{-1} = (\mathcal{F}S\mathcal{F}^{-1})^T$ can be computed in parallel by performing $\lceil \log T \rceil$ sequential squarings for each element along the principal diagonal of $\mathcal{F}S\mathcal{F}^{-1}$ according to the decomposition of Λ^T given earlier.

[**Step 3. Elementwise Product.**] We compute $\mathcal{F}a_T$ by taking the product of $\mathcal{F}S^T\mathcal{F}^{-1}$ and $\mathcal{F}a_0$. As in step 2, every element of $\mathcal{F}a_T = (\mathcal{F}S^T\mathcal{F}^{-1})(\mathcal{F}a_0)$ can be computed in parallel, since we are multiplying a vector by a diagonal matrix.

[**Step 4. Inverse FFT.**] We now compute a_T by applying an inverse FFT to $\mathcal{F}a_T$.

THEOREM 4.1. *STENCILFFT-P computes the Tth timestep of a stencil computation on a periodic grid of N cells in $\Theta(N \log(NT))$ work and $\Theta(\log T + \log N \log \log N)$ span.*

PROOF. Theorem follows from bounds given in Table 3. □

Table 3. Work and Span Complexity Bounds for Steps 1–4 of StencilFFT-P

Step(s)	T_1	T_∞
1,4	$\Theta(N \log N)$	$\Theta(\log N \log\log N)$
2	$\Theta(N \log T)$	$\Theta(\log N + \log T)$
3	$\Theta(N)$	$\Theta(\log N)$

Note that we include thread spawning time from the Binary Forking model in these bounds.

4.2 Generalizations

We now give overviews of some straightforward generalizations of the 1D algorithm given above. These greatly enhance the scope of stencil problems our algorithm can handle, yet do not require significant conceptual work beyond what we have already presented. There are no changes in the asymptotic complexities of our algorithm when any of these generalizations are applied.

Multi-Dimensional Stencils. The first extension of the 1D version of our algorithm is to support arbitrarily high-dimensional grids. The notions involved with solving across a grid of size $\ell_1 \times \cdots \times \ell_d = N$ differ from a 1D grid only in that we now require indices for every grid dimension.

Although the stencil used and grid data used are indexed by an integer, they will now be indexed by a d-dimensional vector $\boldsymbol{i} = [i_1, \ldots, i_d]$. Let $a_0[\boldsymbol{i}]$ be the initial data for a grid of size $\ell_1 \times \cdots \times \ell_d = N$ to be acted on by stencil $S = \sum_{\boldsymbol{i}} S[\boldsymbol{i}, \boldsymbol{0}]X^{\boldsymbol{i}}$, where $X^{\boldsymbol{i}} = X_1^{i_1} \cdots X_d^{i_d}$ is a shift operator whose action is given by $(X^{\boldsymbol{k}} a)[\boldsymbol{i}] = a[\boldsymbol{i} - \boldsymbol{k}]$ and whose matrix elements are $X^{\boldsymbol{k}}[\boldsymbol{i}, \boldsymbol{j}] = \delta_{\boldsymbol{i}, \boldsymbol{j}+\boldsymbol{k}}$. We have periodic boundary conditions, so X_j can be viewed as cyclically permuting the jth dimension. Future timesteps are defined with $a_{t+1} = Sa_t = S^{t+1}a_0$, where matrix multiplications are contractions over the internal vector indices, $(Sa)[\boldsymbol{i}] = \sum_{\boldsymbol{j}} S[\boldsymbol{i}, \boldsymbol{j}]a[\boldsymbol{j}]$.

As with the 1D case, S is a circulant matrix, so we can apply DFTs over every spatial dimension to diagonalize it. These DFT matrices together make up an operator with elements given by $\mathcal{F}[\boldsymbol{i}, \boldsymbol{j}] = \omega_{\ell_1}^{-i_1 j_1} \cdots \omega_{\ell_d}^{-i_d j_d}$ and $\mathcal{F}^{-1}[\boldsymbol{i}, \boldsymbol{j}] = \omega_{\ell_1}^{i_1 j_1} \cdots \omega_{\ell_d}^{i_d j_d}/(\ell_1 \cdots \ell_d)$. Note of course that by "diagonal" here we mean that $\Lambda[\boldsymbol{i}, \boldsymbol{j}] = (\mathcal{F}S\mathcal{F}^{-1})[\boldsymbol{i}, \boldsymbol{j}] = 0$ for all $\boldsymbol{i} \neq \boldsymbol{j}$. The proof that Λ can be found by applying a single Multi-FFT (see Figure 6) over the first column of S is made by expanding the diagonal matrix of eigenvalues $\mathcal{F}A\mathcal{F}^{-1} = \Lambda$ directly gives us

$$\begin{aligned}
\Lambda[i,i] = (\mathcal{F}A\mathcal{F}^{-1})[i,i] &= \left(\mathcal{F}\left(\sum_j A[j,0]X^j\right)\mathcal{F}^{-1}\right)[i,i] \\
&= \sum_j A[j,0](\mathcal{F}X^j\mathcal{F}^{-1})[i,i] \\
&= \sum_j A[j,0]\sum_k\sum_\ell \omega_N^{-i\ell}\delta_{\ell,k+j}\omega_N^{ki}/N \\
&= \sum_j A[j,0]\sum_k \omega_N^{-i(k+j)}\omega_N^{ki}/N = \sum_j \omega_N^{-ij}A[j,0] \\
&= (\mathcal{F}A)[i,0] = (\mathcal{F}a)[i],
\end{aligned}$$

where in the last line a is the first column of A. A is any circulant matrix, so the same raltion applies to S.

Switching from 1D to d-dimensional results in the version of the StencilFFT-P algorithm whose pseudocode given in Figure 6. Since Multi-FFT is of equivalent work, span, and serial cache complexity as a standard FFT [47], this version of the StencilFFT-P algorithm is of identical complexity to that already shown in Theorem 4.1, so long as we assume that $d = \Theta(1)$ for our complexity analysis.

Implicit Stencils. The second way we expand the set of problems our algorithm can handle is by giving a method for handling implicit stencils [80]. This is accomplished by mapping them to mathematically equivalent explicit stencils.

We extend StencilFFT-P to handle stencils that depend implicitly on field data from the current timestep by computing a pseudoinverse after diagonalizing the implicit part of the stencil.

Let S and Q be explicit stencils, and suppose we wish to solve a stencil problem with update equation $Qa_{t+1} = Sa_t$. First, we diagonalize $\mathcal{F}Q\mathcal{F}^{-1} = \Lambda_Q$ and $\mathcal{F}S\mathcal{F}^{-1} = \Lambda_S$ by pushing them to the frequency domain, after which we will take the pseudoinverse of Λ_Q, defined as $\Lambda_Q^+[i,i] = \Lambda_Q[i,i]^{-1}$ if $\Lambda_Q[i,i] \neq 0$ and 0 otherwise. Now we find a new diagonalized stencil $\Lambda_R = \Lambda_Q^+ \Lambda_S$. This new stencil R has the following property: If $a_{t+1} = Ra_t$, then $Qa_{t+1} = Sa_t$.

Thus we can evolve data according to R, and the evolved data will satisfy the original implicit stencil equation. Supporting implicit stencils is a significant capability in a stencil solver, as implicit stencils can often be designed to be more stable than explicit ones.

Vector-Valued Fields. The third way we extend our algorithms is to handle vector-valued fields, i.e., fields where the grid data for each cell is an array of fixed length instead of being just a scalar. Consider a set of scalar-valued fields $\{a_t^{(i)}\}$ that evolve according to linear stencils as

$$a_{t+1}^{(i)} = \sum_j S^{(i,j)} a_t^{(j)},$$

where $S^{(i,j)}$ is a circulant stencil matrix describing how $a_{t+1}^{(i)}$ is dependent on $a_t^{(j)}$. As in the scalar-field case, we can diagonalize all the stencils $S^{(i,j)}$ by moving them to the frequency domain.

Since we now have more than one stencil, we have to revisit our treatment of sequential squaring. Let $\{R^{(i,j)}\}$ be a set of stencils that evolves a_t forward r timesteps to a_{t+r},

$$a_{t+r}^{(i)} = \sum_j R^{(i,j)} a_t^{(j)}.$$

We can find a new set of stencils $\{Q^{(i,j)}\}$ that evolve data forward $2r$ timesteps by reading them off of

$$\sum_j Q^{(i,k)} a_t^{(k)} = a_{t+2r}^{(i)} = \sum_j R^{(i,j)} a_{t+r}^{(j)} = \sum_j R^{(i,j)} \sum_k R^{(j,k)} a_t^{(k)}.$$

This gives the pleasing result that

$$Q^{(i,k)} = \sum_j R^{(i,j)} R^{(j,k)},$$

so all we have to do to support vector-valued fields is to swap out our sequential squaring of Λ with a sequential squaring of a matrix of stencils.

Furthermore, when there are only a constant number of scalar fields $\{a_t^{(i)}\}$ that make up our grid data, the computational complexity of finding $\{Q^{(i,j)}\}$ differs only by a constant factor from squaring Λ in our scalar-field version of the algorithm.

As an example of what can be done with vector-valued fields, suppose we want to implement an affine stencil $Aa_t = Sa_t + c$ on some originally scalar field, where S is a linear stencil and c is a constant. We could then add a spatial field (making the underlying data consist of vectors with 2 elements each) and define $a_0^{(1)} = c$, $S^{(1,1)} = S^{(0,1)} = I$, $S^{(1,0)} = 0$, and $S^{(0,0)} = A$. This realized the behavior of the affine stencil on $a_t^{(0)}$.

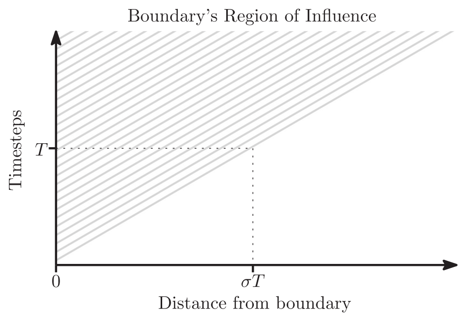

Fig. 7. The growth of the boundary's region of influence over time. Here we assume a stencil that uses σ cells in each direction.

It is noteworthy that if there are a constant number of scalar fields $a_t^{(i)}$ making up our grid data, then the complexity of finding the final data is only a constant factor greater than if the field were scalar.

Vector-valued grid data allows us to support *much larger classes of stencils*, such those that are affine or require data from multiple prior timesteps.

5 APERIODIC STENCIL ALGORITHM

In this section, we will first consider how introducing aperiodicity into our boundary conditions leads to changes in the final data and then present an algorithm for computing the values of the specific cells that are affected by the aperiodic boundary conditions.

5.1 The Effect of Aperiodicity

Often numerical computations require boundary conditions that are not periodic [15, 20, 73, 98, 129], including well-known classes such as Dirichlet [145] and Von Neumann [90], as well as more exotic options [125]. Indeed, the set of potential aperiodic boundary conditions is extremely diverse. Here we will not attempt to describe *how* given aperiodic boundary conditions change the final data but rather *what sections* of the final data are changed.

The set of cells that are dependent on values from the grid boundary are called the boundary's region of influence [13, 35, 96, 139]. It is common for the boundary's region of influence to be smaller than the entire spatial grid, as would be the case in a simulation of a large spatial region for a small time period. A cell's influence on its neighbors is mediated by the stencil, so the larger the stencil is the fewer timesteps it takes for one cell to influence the whole grid.

Consider a stencil with radius σ, i.e., one that only uses values from a neighborhood extending up to σ cells away from its center. After one timestep of a computation based on this stencil, the set of cells that are influenced by aperiodic boundary conditions will be exactly those that are at distance σ or less from the grid boundary. After T timesteps, all cells within distance σT of the boundary will be influenced. This suggests that we should visualize how the boundary's region of influence grows by drawing a diagram in two variables (as in Figure 7), one being time and the other distance from the boundary. Figures of this type will be of significant use to us while describing our aperiodic algorithm.

We now move to our aperiodic algorithm StencilFFT-A, at the core of which is Recursive-Boundary, a divide-and-conquer technique for correcting the values of the cells in the boundary's region of influence.

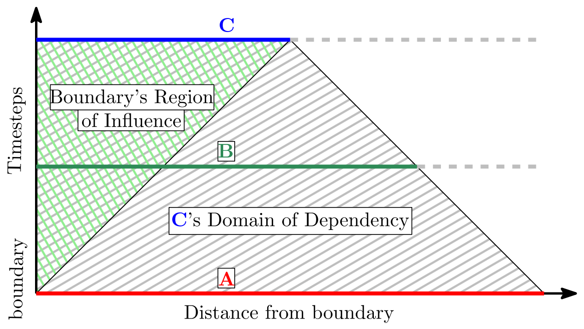

Fig. 8. The set of cells relevant to our divide-and-conquer approach to solving for the boundary's region of influence. We start with set A, from which we will compute set B as an intermediate step, after which we will solve for C.

5.2 Correcting Final Values in the Boundary's Region of Influence

Our aperiodic algorithm breaks up the final data into two regions that will be computed with different methods. The *interior* region consists of cells whose values will not change no matter what boundary conditions we apply. In particular, periodic boundary conditions would not change the value of these cells. This region will thus be dealt with by using our efficient periodic algorithm. The *exterior* region is the boundary's region of influence, made up of cells that are dependent on boundary values.

Since we already know how to solve for the interior region, let us turn our attention to the boundary's region of influence, which we solve for with a recursive divide-and-conquer approach. The core idea here will be to perform a time-cut to reduce the number of cells affected by the aperiodic boundary; by splitting one large step into two smaller steps we will be able to use the periodic solver on more space at the cost of having to compute cell values at an intermediate time.

Figure 8 diagrammatically shows how RecursiveBoundary solves for the boundary region. First, all values in region A are used to solve for region B, and then region B is used to solve for region C. These regions can be much smaller than the entire spatial grid, since even though they wrap around the entirety of the boundary, they are only a fixed number of cells thick.

Suppose again that we are performing computations with a stencil of radius σ. Then set C is the boundary's region of influence after T timesteps, which includes all cells within distance σT of the boundary. Since we want to be able to fully compute C from B and B from A, we must continue to extend our regions as we go $T/2$ steps back for each; B includes everything within $3\sigma T/2$ cells of the boundary, and A includes everything out to a distance of $2\sigma T$. We now refine our naming of these regions, defining subregions $A_1, A_2, B_1, B_2, B_3, C_1$, and C_2, whose sizes and locations are shown in Figure 9.

The language we have used up to this point has been dimension free, in that every statement made has been independent of the number of dimensions in the spatial grid we are performing computations on. Now, to make clear the regions shown in Figures 8 and 9, we will consider what exactly they look like when we make a choice of spatial grid dimension.

In *1D*, when the spatial grid $a[0, \ldots, N-1]$ is just a linear array of N cells, the boundary lies to the left of $a[0]$ and to the right of $a[N-1]$. The set of all cells within distance d of the 1D grid's boundary is given by the union of $a[0, \ldots, d-1]$ and $a[N-d, \ldots, N-1]$. For example, the region C is the union of $a[0, \ldots, \sigma T-1]$ and $a[N-\sigma T, \ldots, N-1]$.

In *2D* things become more complicated, since the boundary of the $n_1 \times n_2$ spatial grid $a[0..(n_1-1), 0..(n_2-1)]$ lies on the outside of the connected set of cells made up of the 1D subarrays

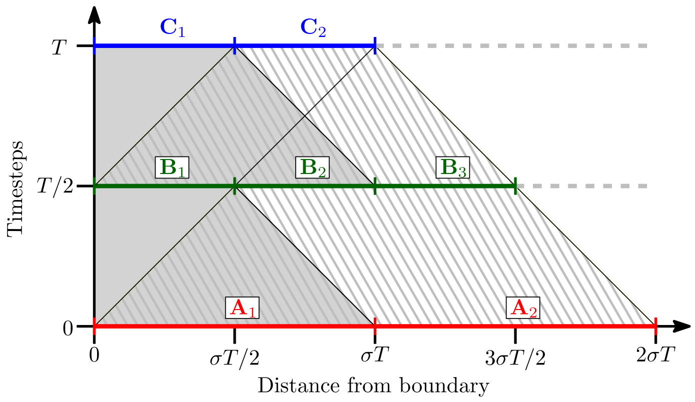

Fig. 9. Additional detail on the regions already specified in Figure 8. The dark regions show areas that will be dealt with as RecursiveBoundary subproblems; the light areas can be handled by StencilFFT-P.

$a[0, 0..(n_2-1)]$, $a[n_1-1, 0..(n_2-1)]$, $a[0..(n_1-1), 0]$, and $a[0..(n_1-1), n_2-1]$. The set of all cells within distance d of the 2D grid's boundary is the union of $a[0..(d-1), 0..(n_2-1)]$, $a[(n_1-d)..(n_1-1), 0..(n_2-1)]$, $a[0..(n_1-1), 0..(d-1)]$, and $a[0..(n_1-1), (n_2-d)..(n_2-1)]$. Note that we are listing some cells twice here, specifically those in the corners of the grid.

Now we are prepared to describe in detail[7] the RecursiveBoundary algorithm for computing the correct values of cells in the boundary's region of influence. Consider again Figure 9. The RecursiveBoundary algorithm solves for two time-slices that are done sequentially. Each time-slice is divided into two distinct regions: one that is solved for with a periodic solver and one that is solved for recursively. These two regions are handled in parallel.

(1) ***Solving for B from A.*** We will begin by feeding data from region $A = A_1 \cup A_2$ into our periodic solver to find values for region $B_2 \cup B_3$. For a d-dimensional grid, this will take $2d$ calls to the periodic solver. At the same time (in parallel), a recursive call is made to the RecursiveBoundary algorithm with A_1 as input, writing the output to B_1. This completes all the values of B.
(2) ***Solving for C from B.*** Now $B = B_1 \cup B_2 \cup B_3$ is fed into our periodic algorithm to find C_2, while in parallel we feed $B_1 \cup B_2$ into another recursive call to RecursiveBoundary, which will find the values for C_1. Thus all cell values in $C = C_1 \cup C_2$ are computed.

The base cases in recursion occur when the only cells to be computed are within a constant distance of the boundary. In this case, a standard looping algorithm is applied. Theoretically, we can set the cutoff distance to any constant greater than or equal to σ, but in practice the constant is chosen such that the cost of computing the base case is balanced with the cost of further recursion. See Figure 11 for pseudocode.

Combining our periodic solver for the interior region with this recursive solver for the boundary region gives our aperiodic algorithm StencilFFT-A. A diagrammatic outline of StencilFFT-A for 1D spatial grid is given in Figure 10, and pseudocode for d dimensions is given in Figure 11.

Theorem 5.1. *The StencilFFT-A algorithm can compute the Tth timestep of a stencil computation on a grid of size N with aperiodic boundary conditions in $\Theta(bT\log(bT)\log T + N\log N)$ work and $\Theta(T\log b + \log N\log\log N)$ span, where b is the number of grid cells defined by boundary conditions.*

[7]For brevity here we use the region names as shown in Figure 9. The reader who would prefer to see these regions specified directly in terms of their distance from the boundary is referred to Section 5.3.

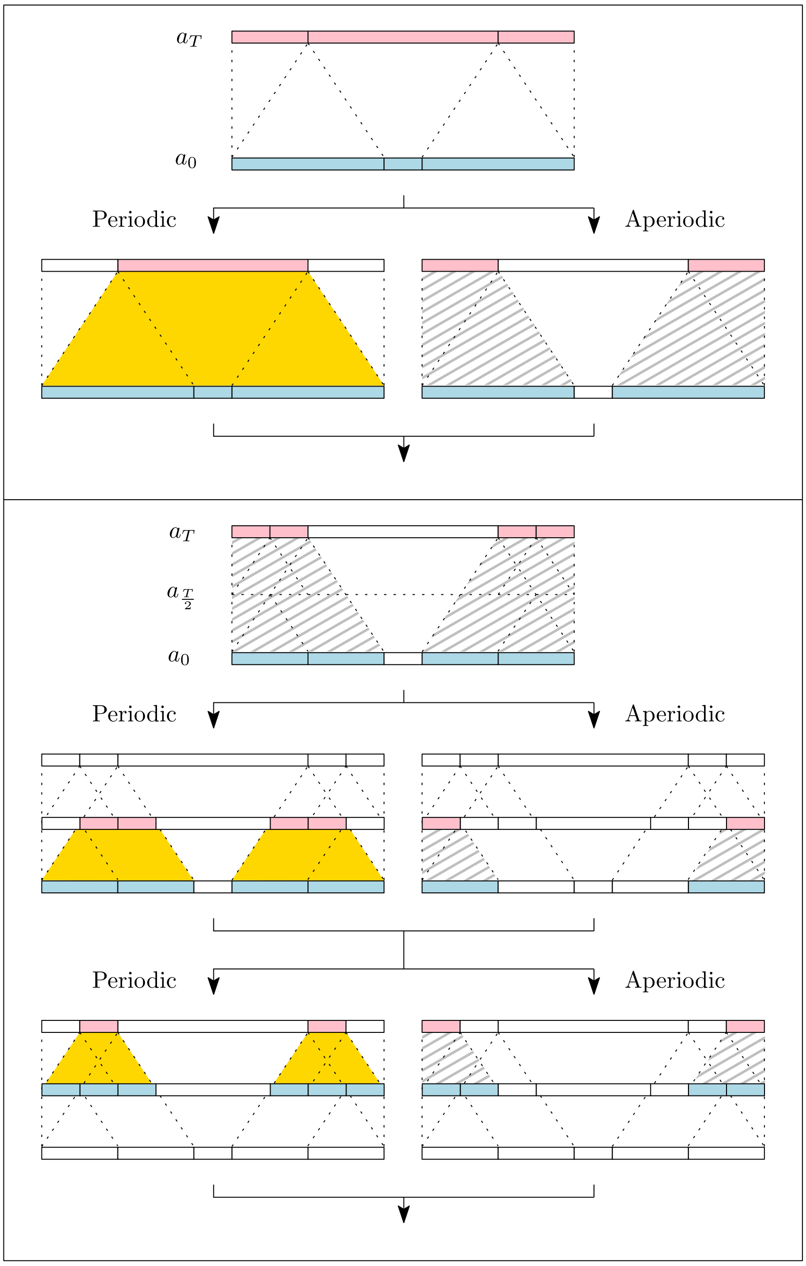

Fig. 10. Block diagram of our FFT-based aperiodic stencil algorithm for a 1D spatial grid.

Proof. The complexities for StencilFFT-P have already been given in Theorem 4.1, so here we need only derive the complexities for the RecursiveBoundary computation algorithm.

Let the spatial grid be of size $\ell_1 \times \cdots \times \ell_d = N$, with $d = \Theta(1)$. We bound the number of cells in the boundary's region of interest by $b\sigma T$, where $b = 2N(\ell_1^{-1} + \cdots + \ell_d^{-1}) = \Omega(N^{1-1/d})$ is the size of the spatial grid's boundary and $\sigma = \Theta(1)$ is the size of the stencil being applied.
[Work.] At every stage of the divide-and-conquer algorithm, we make two recursive calls and

```
StencilFFT-A($s, \sigma, a_0, \ell_0, \ldots, \ell_d, T$)
(1) $\Delta \leftarrow \sigma T/2$
(2) result $\leftarrow$ Array of size $\ell_1 \times \cdots \times \ell_d$
(3) parallel (1):                                                    ▷ Interior
(4)     center $\leftarrow$ StencilFFT-P($s, a_0, \ell_0, \ldots, \ell_d, T$)
(5)     result[$2\Delta <$ dist] $\leftarrow$ center[$2\Delta <$ dist]
(6) parallel (2):                                                    ▷ Boundary
(7)     boundary $\leftarrow$ RecursiveBoundary($s, \sigma, a_0, \ell_0, \ldots, \ell_d, T$)
(8)     result[$0 <$ dist $\leq 2\Delta$] $\leftarrow$ boundary
```

```
RecursiveBoundary($s, \sigma, a_0, \ell_0, \ldots, \ell_d, T$)
(1) $\Delta \leftarrow \sigma T/2$
    [Base Case.] ____________________________________________
(2) if $T <$ cutoff then
(3)     Iteratively solve for cells within $2\Delta$ of the boundary at time $T$
(4)     return
    [Step 1.] _______________________________________________
(5) parallel (1):                                                    ▷ Interior
(6)     $A_{12} \leftarrow a_0[0 <$ dist $\leq 4\Delta]$
(7)     $B_{23} \leftarrow$ StencilFFT-P($s, A_{12}, \ell_1, \ldots, \ell_d, T/2$)
(8)     result[$\Delta <$ dist $\leq 3\Delta$] $\leftarrow B_{23}[\Delta <$ dist $\leq 3\Delta]$
(9) parallel (2):                                                    ▷ Boundary
(10)    $A_1 \leftarrow a_0[0 <$ dist $< 2\Delta]$
(11)    $B_1 \leftarrow$ RecursiveBoundary($s, A_1, \ell_1, \ldots, \ell_d, T/2$)
(12)    result[$0 <$ dist $\leq \Delta$] $\leftarrow B_1$
(13) $B_{123} \leftarrow$ result[$0 <$ dist $\leq 3\Delta$]
    [Step 2.] _______________________________________________
(14) parallel (1):                                                   ▷ Interior
(15)    $C_2 \leftarrow$ StencilFFT-P($s, B_{123}, \ell_1, \ldots, \ell_d, T/2$)
(16)    result[$\Delta <$ dist $\leq 2\Delta$] $\leftarrow C_2[\Delta <$ dist $\leq 2\Delta]$
(17) parallel (2):                                                   ▷ Boundary
(18)    $B_{12} \leftarrow B[0 <$ dist $\leq 2\Delta]$
(19)    $C_1 \leftarrow$ RecursiveBoundary($s, B_{12}, \ell_1, \ldots, \ell_d, T/2$)
(20)    result[$0 <$ dist $\leq \Delta$] $\leftarrow C_1$
(21) $C_{12} \leftarrow$ result[$0 <$ dist $\leq 2\Delta$]
```

Fig. 11. The StencilFFT-A algorithm and the RecursiveBoundary subroutine. The **parallel** keyword is used here to mark blocks of code that are run on separate processes. Throughout both listings we use a dimension-free indexing notation where $A[a < \textbf{dist} \leq b]$ represents the set of cells in A that have distance to the boundary in the range $(a, b]$.

apply the periodic solver to $\Theta(bT)$ grid cells. This gives us the recurrence

$$T_1(T) = \begin{cases} \Theta(b) & T < c, \\ 2T_1(T/2) + \Theta(bT \log(bT)) & T \geq c, \end{cases}$$

for some positive constant c. This gives $T_1(T) = \Theta(bT \log(bT) \log T)$. Taking into account the $\Theta(N \log(TN))$ work that will be done by the periodic solver for the interior region gives a final work bound of $\Theta(bT \log(bT) \log T + N \log N)$.
[Span.] At every level of recursion, both calls to RecursiveBoundary are done in sequence, but in parallel with the $\Theta(1)$ associated periodic solver calls. The recurrence for span is thus

$$T_\infty(T) = \max\{2T_\infty(T/2), c' \log T \log\log T\} + \Theta(\log b),$$

Table 4. Experimental Setup on the Stampede2 Supercomputer [5] using Knights Landing (KNL) Intel Xeon Phi 7250 and Skylake (SKX) Intel Xeon Platinum 8160 nodes

KNL	Cores	68 cores per socket, 1 socket (total: 68 threads)
	Cache sizes	L1 32 KB, L2 1 MB, L3 16 GB (shared)
	Memory	96 GB DDR RAM
SKX	Cores	24 cores per socket, 2 sockets (total: 48 cores)
	Cache sizes	L1 32 KB, L2 1 MB, L3 33 MB
	Memory	144GB /tmp partition on a 200GB SSD
Compiler		Intel C++ Compiler (ICC) v18.0.2
Compiler flags		`-O3 -xhost -ansi-alias -ipo -AVX512`
Parallelization		OpenMP 5.0
Thread affinity		`GOMP_CPU_AFFINITY`

where $c' = \Theta(1)$. This gives $T_\infty(T) = \Theta(T \log b)$, and incorporating the $\Theta(\log T + \log N \log\log N)$ span from the periodic solver across the interior region gives a final bound of $\Theta(T \log b + \log N \log\log N))$. □

Near-Optimality of Bounds. We believe that the bounds we achieve here for work and span are near-optimal (within a polylogarithmic factor) for fully general aperiodic stencil problems. This is because specific choices of nonlinear boundary conditions in combination with linear stencils can result in arbitrary cellular automata being embedded into the boundary of the spatial grid. These automata can be computationally universal (any computation can be mapped to them in a way that preserves time and space complexity), and hence the work complexity will be $\Omega\left(TN^{1-1/d}\right)$ (size of the space-time boundary) and the span complexity will be $\Omega(T)$. However, there may be significant subcases of aperiodic boundary conditions that allow for more efficient computations, such as for the heat equation with Dirichlet boundary conditions.

5.3 ℓ-shell Spatial Grid Decomposition

So far, we have assumed that stencils with some radius σ use the values of cells at distance σ in any particular direction. However, this is not necessarily the case; one could imagine a stencil that requires a large number of values along one dimension and only a few along another. Upwind stencils also break this pattern, requiring values from very asymmetric neighborhoods of cells.

The concept of region of influence used in our derivation of RecursiveBoundary is itself a sufficient basis for defining the regions $A_{1,2}, B_{1,2,3}$, and $C_{1,2}$ shown in Figure 9. Let us define an ℓ-shell of the boundary of a spatial grid to be the spatial region consisting of all cells that enter the boundary's region of influence after exactly ℓ timesteps. Obviously, there can be only one time ℓ when a cell begins to be influenced by the boundary; the set of all ℓ-shells thus fill the spatial grid without overlapping one another.

To generalize the regions shown in Figure 9 to those for arbitrary stencils, all we have to do is switch the interpretation of the horizontal axis from "distance" to "ℓ" and scale it by setting $\sigma = 1$. This yields an algorithm that is more efficient for upwind schemes and other biased stencils.

6 EXPERIMENTAL RESULTS

In this section, we present the experimental evaluation of our algorithms as compared with the state-of-the-art stencil codes. Our experimental setup is shown in Table 4.

Table 5. Benchmark Problems with the Number of Points in the Corresponding Stencils

Benchmark	`heat1d`	`heat2d`	`seidel2d`	`jacobi2d`	`heat3d`	`19pt3d`
Stencil points (α)	3pts	5pts	9pts	25pts	7pts	19pts
Stencil radius (σ)	1	1	1	2	1	2

Table 6. Numerical Accuracy Comparison between Our Algorithms and Looping Code

Stencil	Grid size	Timesteps	Our algorithm	Looping code
`heat1d`	1,000	10^6	5.71632×10^{-6}	5.71637×10^{-6}
`heat2d`	500×500	2.5×10^5	2.73253×10^{-5}	2.73253×10^{-5}
`heat3d`	$200 \times 200 \times 200$	4×10^4	1.72981×10^{-4}	1.72981×10^{-4}

Our analytical solutions were chosen so that the truth values fell within [0.5, 2] everywhere in the solution domain.

Benchmarks and Numerical Accuracy. For benchmarks, we use a variety of stencil problems including those with periodic and aperiodic boundary conditions and across one, two, and three dimensions. Our test stencils, primarily drawn from Reference [112] and listed with details in Table 5, are the following: `heat1d`, `heat2d`, `seidel2d`, `jacobi2d`, `heat3d`, and `19pt3d` (called `poisson3d` in Reference [112]). We test two primary aspects of our algorithms: numerical accuracy and computational complexity.

To evaluate numerical accuracy, we use max relative error against analytical solutions for the heat equation in one, two, and three dimensions. This is shown in Table 6 against a naïve iterative looping implementations that is numerically equivalent to PLuTo. We see that our algorithms show no significant difference in loss from floating point accuracy when compared against standard looping codes.

PLuTo-generated Stencil Programs. The tiled looping implementations were generated by PLuTo [3]—the state-of-the-art tiled looping code generator. The two main types of tiling methods used for performance comparison are `standard` and `diamond`, whose tiles have the shapes of parallelograms and diamonds, respectively. In the plots, we use diamond and square symbols to denote `diamond` and `standard`, respectively. These parallel implementations were run on 68-core KNL and 48-core SKX nodes. The tile sizes were selected via an autotuning phase, exploring sizes from $\{8, 16, 32\}$ for the outer dimensions and from $\{64, 128, 512\}$ for the inner-most dimension to ensure that enough vectorization and multithreaded parallelism were exposed by PLuTo, while ensuring the tile footprint neared the cache size.

Our FFT-based Stencil Programs. Implementations of our FFT-based algorithms use FFT implementations available in the Intel Math Kernel Library (Intel MKL) [1]. In the plots, we use the triangle symbol to represent our FFT-based implementations. The base case sizes used for `1d`, `2d`, `3d` are 128, 64×64, $16 \times 16 \times 16$, respectively.

6.1 Periodic Stencil Algorithms

Figure 12 shows runtime plots for our FFT-based periodic stencil algorithm and PLuTo on Intel KNL nodes and SKX nodes with Figure 13 showing speedup (w.r.t. PLuTo) and scaling plots in respective machines. We identify the implementations of our algorithms in the plots by prefixing the stencil name with "FFT" while PLuTo-generated implementations are identified by a "PLuTo" prefix.

For 1D, 2D, and 3D stencils we keep the value of N fixed to 1.6M, $8K \times 8K$, and $800 \times 800 \times 800$, respectively, and vary T, where $1\text{M} = 10^6$ and $1\text{K} = 10^3$.

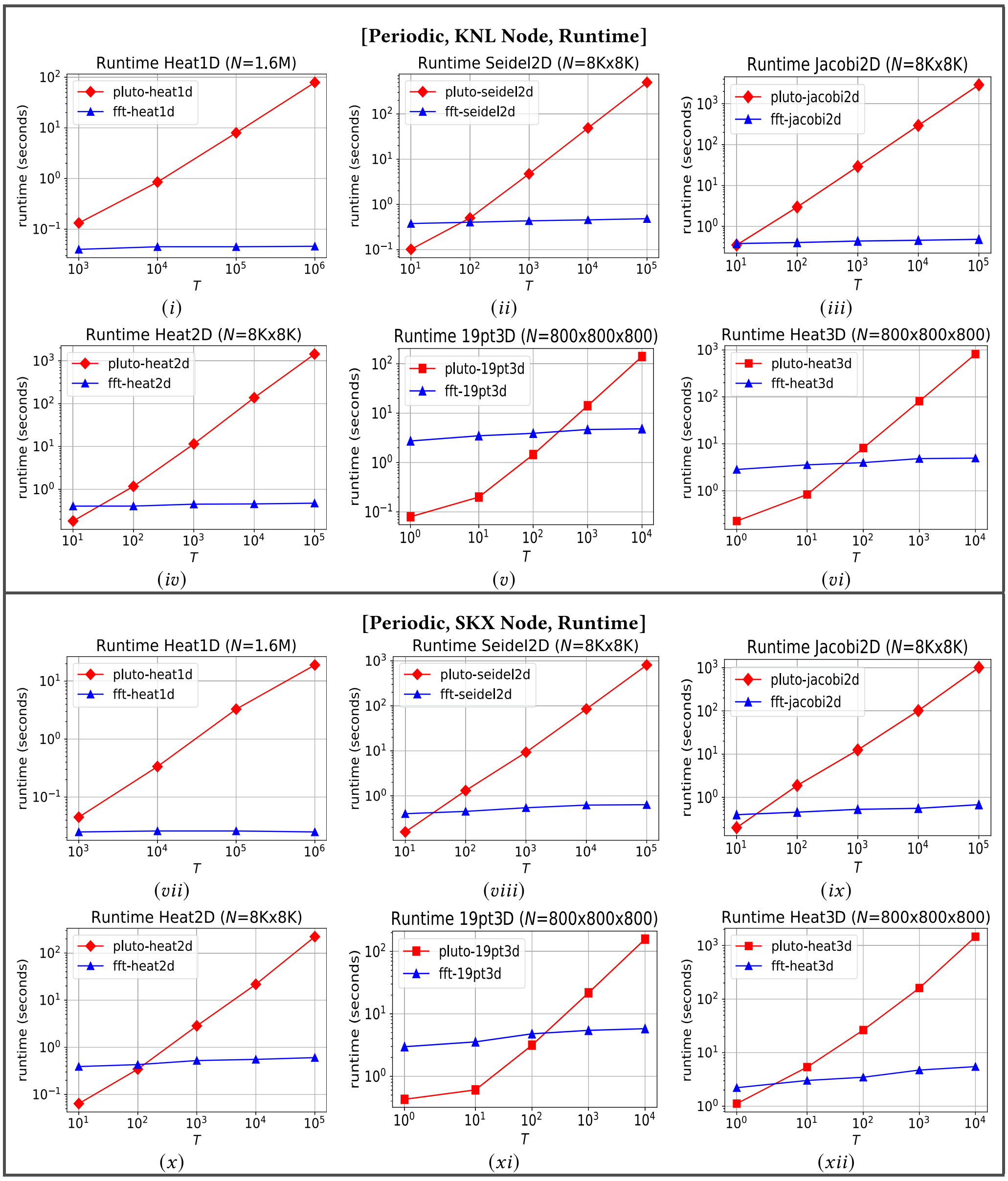

Fig. 12. Performance comparison of our FFT-based *periodic* algorithms with the existing best stencil programs.

In all of our 1D and 2D periodic stencil experiments, diamond ran faster than standard, while in the case of 3D standard outperformed diamond. When N is fixed, performance of our algorithm improved over PLuTo's as T increased, significantly outperforming PLuTo for large T, e.g., for seidel2d our algorithm ran around 6000× faster on KNL when $T = 10^5$. This increase in speedup with the increase of T follows from theoretical predictions. Indeed, theoretical speedup of our algorithm over any existing stencil algorithms is $\Theta(T/\log T)$ when N is fixed (see Table 1). Hence,

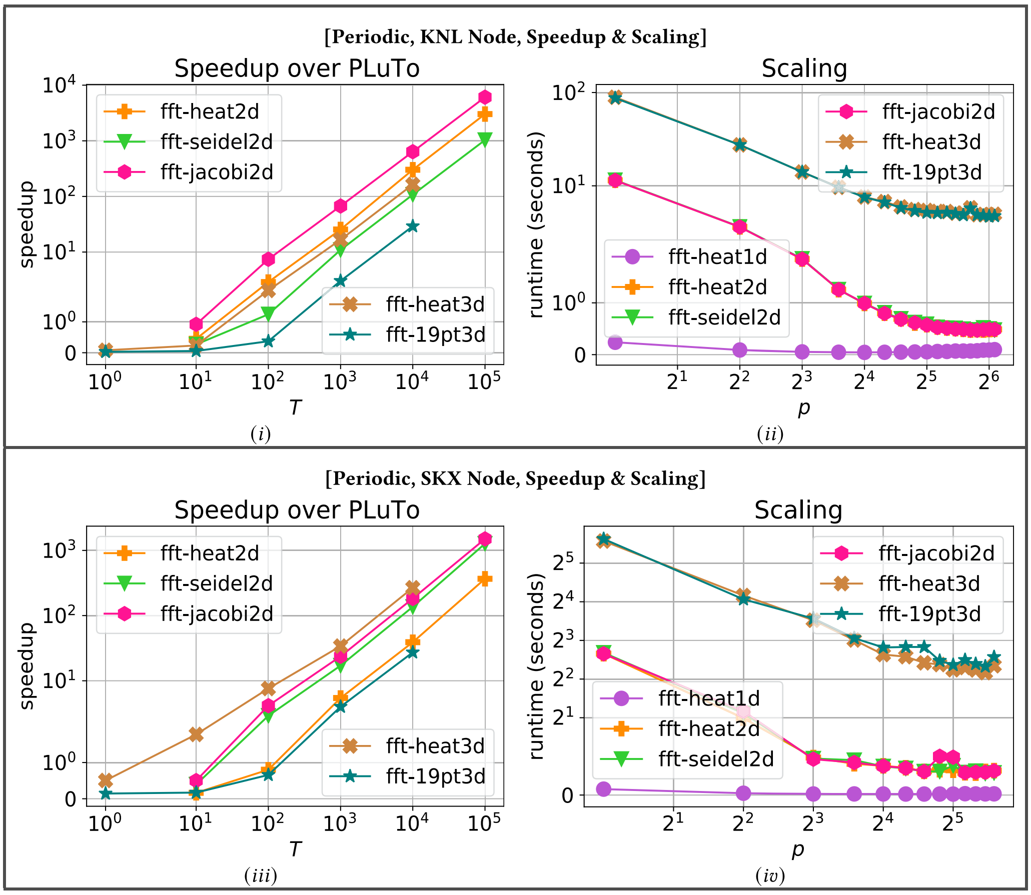

Fig. 13. Performance comparison of our FFT-based *periodic* algorithms with the existing best stencil programs.

the speedup of our algorithm w.r.t. PLuTo-generated codes increased almost linearly with T when N was kept fixed.

Figures 13(ii) and 13(iv) show the scalability of our algorithm on KNL and SKX nodes, respectively, when the number of threads is varied. Our implementations are highly parallel and should scale accordingly. However, we use FFT computations that are memory bound—they perform only $\Theta(N \log N)$ work on an input of size $\Theta(N)$ and thus have very little data reuse.[8] We believe that as a result of this issue, our programs do not scale well beyond 32 threads on KNL and 16 threads on SKX. Indeed, we observe that the FFT and the inverse FFT computations are the scalability bottlenecks of our algorithms.

6.2 Aperiodic Stencil Algorithms

We performed two types of experiments for aperiodic stencils: (1) grid size N was kept fixed while time T was varied, and (2) grid size was set to $N^{1/d} \times \cdots \times N^{1/d}$ and $T = N^{1/d}$ for d dimensions, and N was varied.

[8]Which is evident from their (optimal) $O\left((N/B)\log_M N\right)$ cache complexity with a low temporal locality.

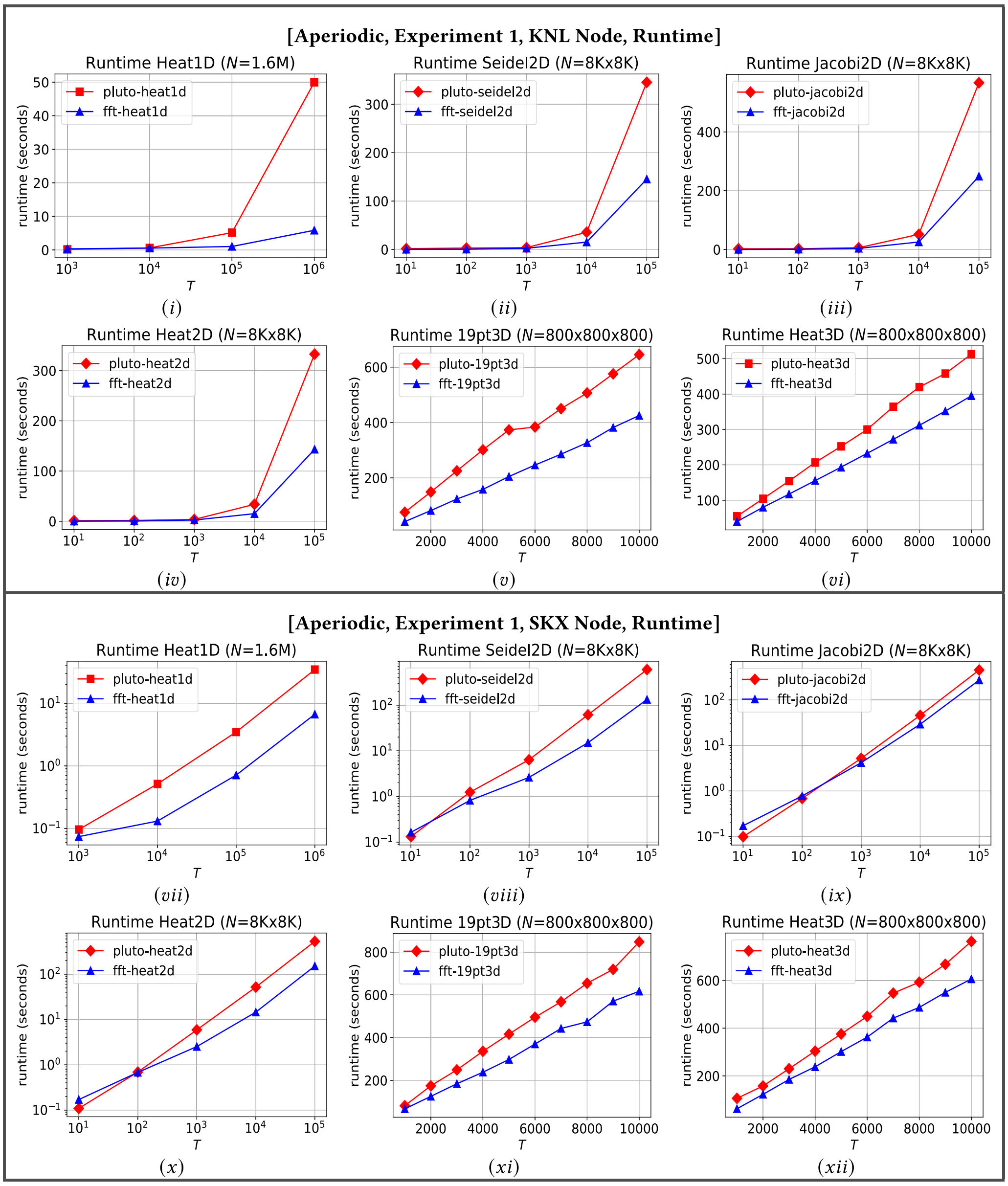

Fig. 14. Performance comparison of our FFT-based *aperiodic* algorithms with the existing best stencil programs.

Figure 14 shows the runtime (w.r.t. PLuTo) plots of our aperiodic stencil algorithm on for experiments 1 on KNL and SKX, respectively. Figures 15 show the corresponding plots for Experiments 2.

In Experiment 1, `diamond` outperformed `standard` for all stencils except for `heat1d` on both machines. Our algorithm always ran faster than PLuTo-generated code, reaching speedup factors of 8.5, 2.3–2.4, and 1.3–1.5 for 1D, 2D, and 3D stencils, respectively, on KNL (see Table 2 for details). The corresponding figures on SKX were 5.2, 1.7–4.5, and 1.3–1.4, respectively. Theoretical bounds in Table 1 imply that our algorithm will run around $\Theta\left(N^{1/d}/(\log TN^{1-1/d})\right)\log T$ factor faster than

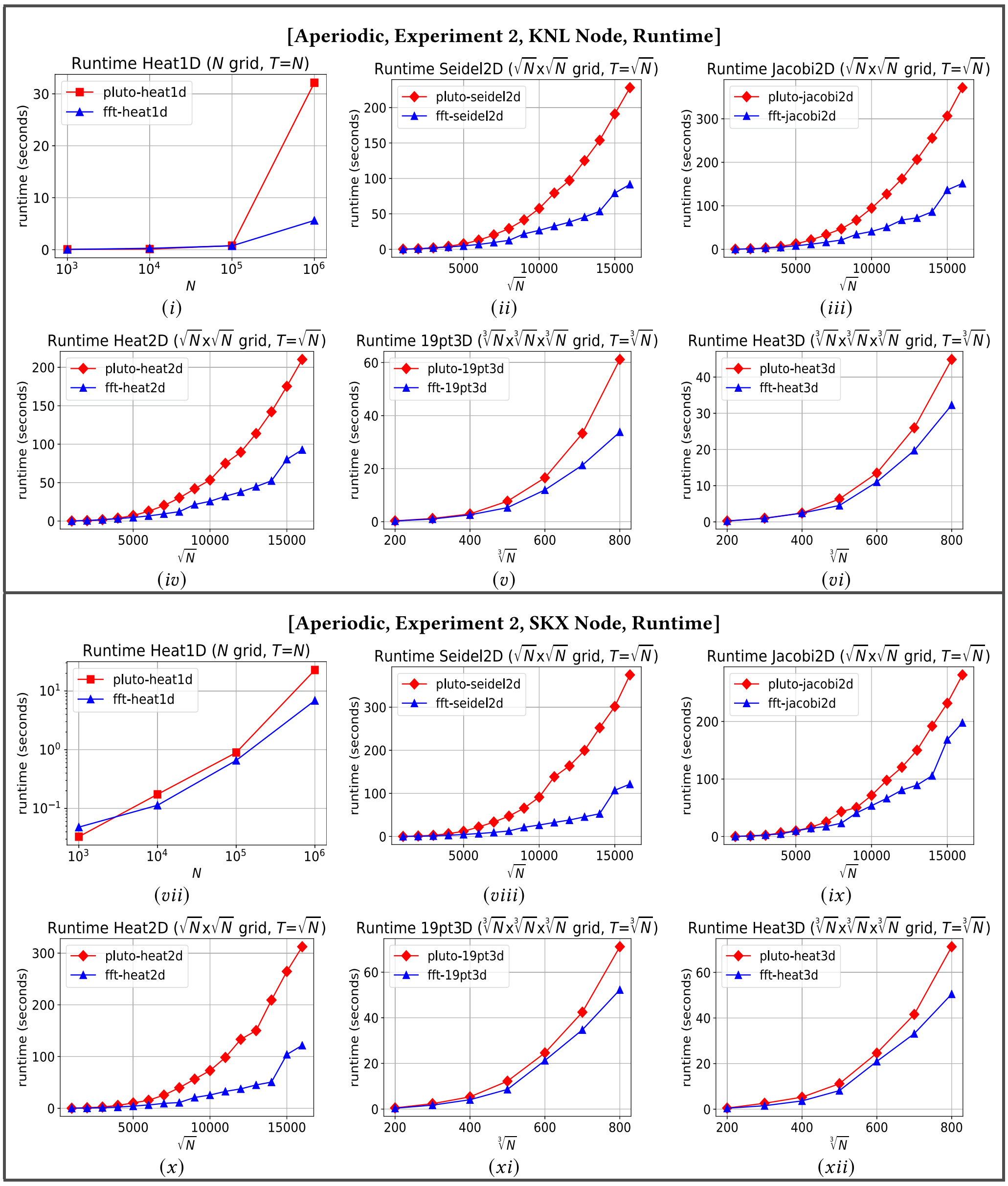

Fig. 15. Performance comparison of our FFT-based *aperiodic* algorithms with the existing best stencil programs.

PLuTo code for any given N and T. So, for a fixed N, the speedup factor will not increase (may even slightly decrease) with the increase of T. The speedup plots match this prediction.

In Experiment 2, `diamond` ran faster than `standard` for all stencils except for `heat1d` and `heat3d` on KNL and `heat1d` on SKX. Our algorithm ran up to 5.7, 2.3–2.5, and 1.4–1.8 factor faster than PLuTo-generated code for 1D, 2D, and 3D stencils, respectively, on KNL (see Table 2). The corresponding speedup factors on SKX were 3.3, 1.4–2.6, and 1.4, respectively. Our theoretical prediction

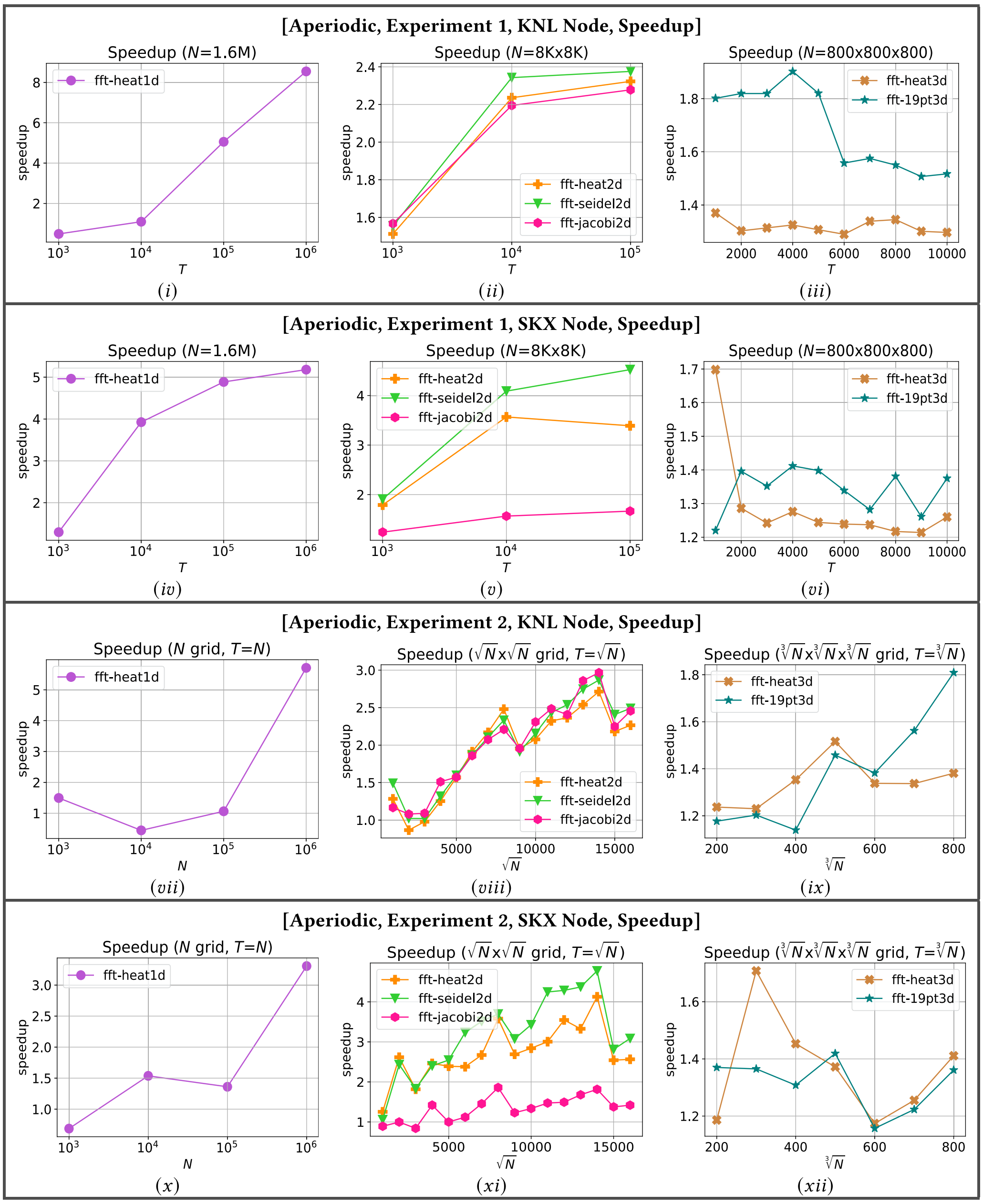

Fig. 16. Performance comparison of our FFT-based *aperiodic* algorithms with the existing best stencil programs.

for rough speedup factor from the previous paragraph implies that our speedup over PLuTo code will increase with the increase of N, which is confirmed by the speedup plots for this experiment. However, the speedup plot of heat2d, seidel2d, and jacobi2d on KNL, as shown in Figure 16(viii) has speedup drops at $N = 9{,}000 \times 9{,}000$ and $N = 15{,}000 \times 15{,}000$. Similar performance drops are also observed on SKX nodes (see Figure 16(xi) in the Appendix). We believe that this happens

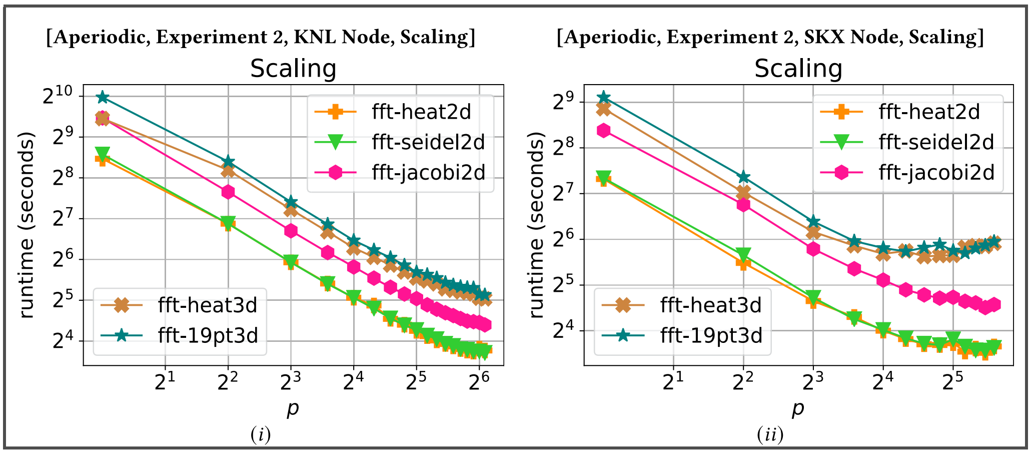

Fig. 17. Scalability of our FFT-based *aperiodic* algorithms for Experiment 2.

mainly because of a known phenomenon that is the drastic performance variations MKL suffers from when the sizes of the spatial grid dimensions change [78]. This performance drop is also partly due to the changes in the base case kernel size of our implementations resulting from the changes in the grid size.

Figure 17 show the scalability plots of our FFT-based aperiodic stencil algorithm on KNL and SKX nodes, respectively. We used $N = 1M$, $N = 16K \times 16K$, and $N = 800 \times 800 \times 800$ for 1D, 2D, and 3D stencils, respectively, and set $T = N^{1/d}$ for our scalability analysis, where d is the number of dimensions. Our implementations show highly scalable performance on KNL and almost similar scalability for 1D and 2D stencils on SKX.

7 CONCLUSION

In this article, we presented a pair of efficient algorithms based on *fast Fourier transforms* for performing *linear* stencil computations with periodic and aperiodic boundary conditions. These are the first high-performing $o(NT)$-work[9] stencil algorithms of significant generality for computing the spatial grid values at the final timestep from the input grid without explicitly generating values for most of the intermediate timesteps. Our stencil algorithms improve computational complexity and parallel running time bounds over the state-of-the-art stencil algorithms by a *polynomial factor*. Experimental results show that implementations of our algorithms run orders of magnitude faster than state-of-the-art implementations for periodic stencils and 1.3× to 8.5× faster for aperiodic stencils, while exhibiting no significant loss in numerical accuracy from floating point arithmetic.

A few interesting problems that one could aim to solve in the future include the following: (1) designing efficient algorithms for certain classes of *nonlinear* stencils and stencils with *conditionals*, (2) designing *low-span* algorithms for aperiodic stencils, and (3) designing algorithms to approximate *inhomogenous* stencils.

ACKNOWLEDGMENTS

We thank Charles E. Leiserson for helpful feedback. The authors also thank anonymous reviewers for their helpful comments to improve the article.

[9] N and T are the spatial grid size and the number of timesteps, respectively.

REFERENCES

[1] Intel MKL. Intel Math Kernel Library. Retrieved from https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html

[2] LLVM MKL. LLVM Framework for High-Level Loop and Data-Locality Optimizations. Retrieved from https://polly.llvm.org/

[3] Pluto. An Sutomatic Parallelizer and Locality Optimizer for Affine Loop Nests. Retrieved from http://pluto-compiler.sourceforge.net/

[4] PoCC. The Polyhedral Compiler Collection. Retrieved from http://web.cs.ucla.edu/~pouchet/software/pocc/

[5] Stampede2. The Stampede2 Supercomputing Cluster. Retrieved from https://www.tacc.utexas.edu/systems/stampede2

[6] Vincent Acary and Bernard Brogliato. 2010. Implicit euler numerical scheme and chattering-free implementation of sliding mode systems. *Syst. Contr. Lett.* 59, 5 (2010), 284–293.

[7] Oliviero Andreussi, Ismaila Dabo, and Nicola Marzari. 2012. Revised self-consistent continuum solvation in electronic-structure calculations. *J. Chem. Phys.* 136, 6 (2012), 064102.

[8] Hafez Asgharzadeh and Iman Borazjani. 2017. A newton–krylov method with an approximate analytical jacobian for implicit solution of navier–stokes equations on staggered overset-curvilinear grids with immersed boundaries. *J. Comput. Phys.* 331 (2017), 227–256.

[9] A. Ashrafizadeh, C. B. Devaud, and N. U. Aydemir. 2015. A jacobian-free newton–krylov method for thermalhydraulics simulations. *Int. J. Numer. Methods Fluids* 77, 10 (2015), 590–615.

[10] Abdon Atangana and Juan Jose Nieto. 2015. Numerical solution for the model of RLC circuit via the fractional derivative without singular kernel. *Adv. Mech. Eng.* 7, 10 (2015), 1687814015613758.

[11] Roni Avissar and Roger A. Pielke. 1989. A parameterization of heterogeneous land surfaces for atmospheric numerical models and its impact on regional meteorology. *Monthly Weather Rev.* 117, 10 (1989), 2113–2136.

[12] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. 2012. Tiling stencil computations to maximize parallelism. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* 1–11.

[13] Stanley L. Barnes. 1964. A technique for maximizing details in numerical weather map analysis. *J. Appl. Meteorol. Climatol.* 3, 4 (1964), 396–409.

[14] Timothy J. Barth and Herman Deconinck. 2013. *High-order Methods for Computational Physics.* Vol. 9. Springer Science & Business Media.

[15] John H. Beggs, Raymond J. Luebbers, Kane S. Yee, and Karl S. Kunz. 1992. Finite-difference time-domain implementation of surface impedance boundary conditions. *IEEE Trans. Antennas Propagat.* 40, 1 (1992), 49–56.

[16] Michael A. Bender, Erik D. Demaine, Roozbeh Ebrahimi, Jeremy T. Fineman, Rob Johnson, Andrea Lincoln, Jayson Lynch, and Samuel McCauley. 2016. Cache-adaptive analysis. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures.* 135–144.

[17] Michael A. Bender, Roozbeh Ebrahimi, Jeremy T. Fineman, Golnaz Ghasemiesfeh, Rob Johnson, and Samuel McCauley. 2014. Cache-adaptive algorithms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms.*

[18] Michele Benzi and Gene H. Golub. 2004. A preconditioner for generalized saddle point problems. *SIAM J. Matrix Anal. Appl.* 26, 1 (2004), 20–41.

[19] Michele Benzi, Michael Ng, Qiang Niu, and Zhen Wang. 2011. A relaxed dimensional factorization preconditioner for the incompressible navier–stokes equations. *J. Comput. Phys.* 230, 16 (2011), 6185–6202.

[20] Stefan Bilbao. 2013. Modeling of complex geometries and boundary conditions in finite difference/finite volume time domain room acoustics simulation. *IEEE Trans. Aud. Speech Lang. Process.* 21, 7 (2013), 1524–1533.

[21] Jiri Blazek. 2015. *Computational Fluid Dynamics: Principles and Applications.* Butterworth-Heinemann.

[22] Guy E. Blelloch, Jeremy T. Fineman, Yan Gu, and Yihan Sun. 2019. Optimal parallel algorithms in the binary-forking model. arXiv:1903.04650. Retrieved from https://arxiv.org/abs/1903.04650

[23] Uday Bondhugula, Aravind Acharya, and Albert Cohen. 2016. The pluto+ algorithm: A practical approach for parallelization and locality optimization of affine loop nests. *ACM Trans. Program. Lang. Syst.* 38, 3 (2016), 1–32.

[24] Uday Bondhugula, Vinayaka Bandishti, and Irshad Pananilath. 2017. Diamond tiling: Tiling techniques to maximize parallelism for stencil computations. *IEEE Trans. Parallel Distrib. Syst.* 28, 5 (2017), 1285–1298.

[25] Uday Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. 2008. Pluto: A practical and fully automatic polyhedral program optimization system. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI'08).* Citeseer.

[26] Rick Borrell, Oriol Lehmkuhl, F. Xavier Trias, and Assensi Oliva. 2011. Parallel direct poisson solver for discretisations with one fourier diagonalisable direction. *J. Comput. Phys.* 230, 12 (2011), 4723–4741.

[27] John P. Boyd. 2001. *Chebyshev and Fourier Spectral Methods.* Courier Corporation.

[28] Ronald Newbold Bracewell and Ronald N. Bracewell. 1986. *The Fourier Transform and its Applications*. Vol. 31999. McGraw–Hill, New York.

[29] Peter N. Brown and Youcef Saad. 1994. Convergence theory of nonlinear newton–krylov algorithms. *SIAM J. Optim.* 4, 2 (1994), 297–330.

[30] Georg Bruun. 1978. z-transform DFT filters and FFT's. *IEEE Trans. Acoust. Speech Sign. Process.* 26, 1 (1978), 56–63.

[31] Raymond H. Chan, James G. Nagy, and Robert J. Plemmons. 1993. FFT-based preconditioners for toeplitz-block least squares problems. *SIAM J. Numer. Anal.* 30, 6 (1993), 1740–1768.

[32] Tony F. Chan. 1984. Stability analysis of finite difference schemes for the advection-diffusion equation. *SIAM J. Numer. Anal.* 21, 2 (1984), 272–284.

[33] Tony F. Chan. 1988. An optimal circulant preconditioner for toeplitz systems. *SIAM J. Sci. Stat. Comput.* 9, 4 (1988), 766–771.

[34] Bang-Fuh Chen and Roger Nokes. 2005. Time-independent finite difference analysis of fully non-linear and viscous fluid sloshing in a rectangular tank. *J. Comput. Phys.* 209, 1 (2005), 47–81.

[35] Evgenii V. Chizhonkov and Maxim A. Olshanskii. 2000. On the domain geometry dependence of the LBB condition. *ESAIM: Math. Model. Numer. Anal.* 34, 5 (2000), 935–951.

[36] Dok Hee Choi and Wolfang J. R. Hoefer. 1986. The finite-difference-time-domain method and its application to eigenvalue problems. *IEEE Trans. Microw. Theory Techn.* 34, 12 (1986), 1464–1470.

[37] Matthias Christen, Olaf Schenk, and Helmar Burkhart. 2011. Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium.* IEEE, 676–687.

[38] James W. Cooley and John W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.* 19, 90 (1965), 297–301.

[39] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms.* MIT Press.

[40] John Crank and Phyllis Nicolson. 1947. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 43. Cambridge University Press, 50–67.

[41] Germund Dahlquist and Åke Björck. 2008. *Numerical Methods in Scientific Computing, Volume I.* SIAM.

[42] Kaushik Datta, Shoaib Kamil, Samuel Williams, Leonid Oliker, John Shalf, and Katherine Yelick. 2009. Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM Rev.* 51, 1 (2009), 129–159.

[43] Claude R. Dietrich and Garry N. Newsam. 1997. Fast and exact simulation of stationary gaussian processes through circulant embedding of the covariance matrix. *SIAM J. Sci. Comput.* 18, 4 (1997), 1088–1107.

[44] Yogi A. Erlangga, Cornelis W. Oosterlee, and Cornelis Vuik. 2006. A novel multigrid based preconditioner for heterogeneous helmholtz problems. *SIAM J. Sci. Comput.* 27, 4 (2006), 1471–1492.

[45] Joel H. Ferziger, Milovan Perić, and Robert L. Street. 2002. *Computational Methods for Fluid Dynamics.* Vol. 3. Springer.

[46] A. S. Fokas. 2000. On the integrability of linear and nonlinear partial differential equations. *J. Math. Phys.* 41, 6 (2000), 4188–4237.

[47] Matteo Frigo and Steven G. Johnson. 2005. The design and implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.

[48] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. 1999. Cache-oblivious algorithms. In *Foundations of Computer Science.* 285–297.

[49] Matteo Frigo and Volker Strumpen. 2005. Cache oblivious stencil computations. In *Proceedings of the 19th Annual International Conference on Supercomputing.* 361–366.

[50] Matteo Frigo and Volker Strumpen. 2005. Cache oblivious stencil computations. In *Proceedings of the International Conference on Supercomputing.* 361–366.

[51] Matteo Frigo and Volker Strumpen. 2009. The cache complexity of multithreaded cache oblivious algorithms. *Theory Comput. Syst.* 45, 2 (2009), 203–233.

[52] Jochen Fritz, Insa Neuweiler, and Wolfgang Nowak. 2009. Application of FFT-based algorithms for large-scale universal kriging problems. *Math. Geosci.* 41, 5 (2009), 509–533.

[53] Andreas Frommer, Kathryn Lund, Daniel B. Szyld, et al. 2017. Block krylov subspace methods for functions of matrices.

[54] Vladimír Fuka. 2015. PoisFFT–A free parallel fast poisson solver. *Appl. Math. Comput.* 267 (2015), 356–364.

[55] Charles F. Gammie, Jonathan C. McKinney, and Gábor Tóth. 2003. HARM: A numerical scheme for general relativistic magnetohydrodynamics. *Astrophys. J.* 589, 1 (2003), 444.

[56] Roberto Garrappa, Igor Moret, and Marina Popolizio. 2015. Solving the time-fractional schrödinger equation by krylov projection methods. *J. Comput. Phys.* 293 (2015), 115–134.

[57] Björn Gmeiner, Tobias Gradl, Francisco Gaspar, and Ulrich Rüde. 2013. Optimization of the multigrid-convergence rate on semi-structured meshes by local fourier analysis. *Comput. Math. Appl.* 65, 4 (2013), 694–711.

[58] Israel Gohberg and Vadim Olshevsky. 1994. Fast algorithms with preprocessing for matrix-vector multiplication problems. *J. Complex.* 10, 4 (1994), 411–427.
[59] Gene H. Golub, James M. Ortega, et al. 1992. *Scientific Computing and Differential Equations: An Introduction to Numerical Methods.* Academic Press.
[60] Irving John Good. 1958. The interaction algorithm and practical Fourier analysis. *J. Roy. Stat. Soc.: Ser. B (Methodol.)* 20, 2 (1958), 361–372.
[61] Robert M. Gray. 2006. *Toeplitz and Circulant Matrices: A Review.* Now Publishers.
[62] Yifei Guan and Igor Novosselov. 2019. Two relaxation time lattice boltzmann method coupled to fast fourier transform poisson solver: Application to electroconvective flow. *J. Comput. Phys.* 397 (2019), 108830.
[63] Martin H. Gutknecht. 2007. A brief introduction to krylov space methods for solving linear systems. In *Frontiers of Computational Science.* Springer, 53–62.
[64] Stefan Guttel, Roel Van Beeumen, Karl Meerbergen, and Wim Michiels. 2014. NLEIGS: A class of fully rational Krylov methods for nonlinear eigenvalue problems. *SIAM J. Sci. Comput.* 36, 6 (2014), A2842–A2864.
[65] Bastian Hagedorn, Larisa Stoltzfus, Michel Steuwer, Sergei Gorlatch, and Christophe Dubach. 2018. High performance stencil code generation with lift. In *Proceedings of the International Symposium on Code Generation and Optimization.* 100–112.
[66] Wei He and Shuzhi Sam Ge. 2011. Robust adaptive boundary control of a vibrating string under unknown time-varying disturbance. *IEEE Trans. Contr. Syst. Technol.* 20, 1 (2011), 48–58.
[67] Tom Henretty, Richard Veras, Franz Franchetti, Louis-Noël Pouchet, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. 2013. A stencil compiler for short-vector simd architectures. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing.* 13–24.
[68] Charles Hirsch. 2007. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics.* Elsevier.
[69] Roger W. Hockney. 1965. A fast direct solution of poisson's equation using fourier analysis. *J. ACM* 12, 1 (1965), 95–113.
[70] Ilse C. F. Ipsen and Carl D. Meyer. 1998. The idea behind krylov methods. *Am. Math. Month.* 105, 10 (1998), 889–899.
[71] Eugene Isaacson and Herbert Bishop Keller. 2012. *Analysis of Numerical Methods.* Courier Corporation.
[72] Panuwat Janpugdee, Prabhakar H. Pathak, Pongsak Mahachoklertwattana, and Robert J. Burkholder. 2006. An accelerated DFT-MoM for the analysis of large finite periodic antenna arrays. *IEEE Trans. Antennas Propagat.* 54, 1 (2006), 279–283.
[73] Hans Johnston and Jian-Guo Liu. 2002. Finite difference schemes for incompressible flow based on local pressure boundary conditions. *J. Comput. Phys.* 180, 1 (2002), 120–154.
[74] Matthias Kabel, Thomas Böhlke, and Matti Schneider. 2014. Efficient fixed point and newton–krylov solvers for FFT-based homogenization of elasticity at large deformations. *Comput. Mech.* 54, 6 (2014), 1497–1514.
[75] E. Kalnay, M. Kanamitsu, and W. E. Baker. 1990. Global numerical weather prediction at the national meteorological center. *Bull. Am. Meteorol. Soc.* 71, 10 (1990), 1410–1428.
[76] Shoaib Kamil, Cy Chan, Leonid Oliker, John Shalf, and Samuel Williams. 2010. An auto-tuning framework for parallel multicore stencil computations. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS'10).* IEEE, 1–12.
[77] Shoaib Kamil, Parry Husbands, Leonid Oliker, John Shalf, and Katherine Yelick. 2005. Impact of modern memory subsystems on cache optimizations for stencil computations. In *Proceedings of the Workshop on Memory System Performance.* 36–43.
[78] Semyon Khokhriakov, Ravi Reddy Manumachu, and Alexey Lastovetsky. 2018. Performance optimization of multi-threaded 2D FFT on multicore processors: Challenges and solution approaches. In *Proceedings of the IEEE 25th International Conference on High Performance Computing Workshops (HiPCW'18).* IEEE. https://doi.org/10.1109/hipcw.2018.8634318
[79] Robert C. Kirby and Lawrence Mitchell. 2018. Solver composition across the PDE/linear algebra barrier. *SIAM J. Sci. Comput.* 40, 1 (2018), C76–C98.
[80] Peter E. Kloeden and Eckhard Platen. 1992. Higher-order implicit strong numerical schemes for stochastic differential equations. *J. Stat. Phys.* 66, 1 (1992), 283–314.
[81] Andrew V. Knyazev. 2001. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput.* 23, 2 (2001), 517–541.
[82] S. S. Komissarov. 2002. Time-dependent, force-free, degenerate electrodynamics. *Mon. Not. Roy. Astron. Soc.* 336, 3 (2002), 759–766.
[83] Martin Kong, Richard Veras, Kevin Stock, Franz Franchetti, Louis-Noël Pouchet, and Ponnuswamy Sadayappan. 2013. When polyhedral transformations meet SIMD code generation. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation.* 127–138.

[84] Tugrul Konuk and Jeffrey Shragge. 2020. Modeling full-wavefield time-varying sea-surface effects on seismic data: A mimetic finite-difference approach. *Geophysics* 85, 2 (2020), T45–T55.

[85] Matthias Korch and Tim Werner. 2020. An in-depth introduction of multi-workgroup tiling for improving the locality of explicit one-step methods for ODE systems with limited access distance on GPUs. *Concurr. Comput.: Pract. Exper.* (2020), e6016.

[86] Phaedon-Stelios Koutsourelakis. 2009. Accurate uncertainty quantification using inaccurate computational models. *SIAM J. Sci. Comput.* 31, 5 (2009), 3274–3300.

[87] Arno B. J. Kuijlaars. 2006. Convergence analysis of Krylov subspace iterations with methods from potential theory. *SIAM Rev.* 48, 1 (2006), 3–40.

[88] Dmitri Kuzmin. 2010. A vertex-based hierarchical slope limiter for p-adaptive discontinuous galerkin methods. *J. Comput. Appl. Math.* 233, 12 (2010), 3077–3085.

[89] Olivier Le Maître and Omar M. Knio. 2010. *Spectral Methods for Uncertainty Quantification: with Applications to Computational Fluid Dynamics*. Springer Science & Business Media.

[90] Wenyuan Liao. 2013. A high-order ADI finite difference scheme for a 3D reaction-diffusion equation with neumann boundary condition. *Numer. Methods Partial Differ. Equ.* 29, 3 (2013), 778–798.

[91] Mathias Louboutin, Michael Lange, Fabio Luporini, Navjot Kukreja, Philipp A. Witte, Felix J. Herrmann, Paulius Velesko, and Gerard J. Gorman. 2019. Devito (v3. 1.0): An embedded domain-specific language for finite differences and geophysical exploration. *Geosci. Model Dev.* 12, 3 (2019), 1165–1187.

[92] Fabio Luporini, Mathias Louboutin, Michael Lange, Navjot Kukreja, Philipp Witte, Jan Hückelheim, Charles Yount, Paul H. J. Kelly, Felix J. Herrmann, and Gerard J. Gorman. 2020. Architecture and performance of devito, a system for automated stencil computation. *ACM Trans. Math. Softw.* 46, 1 (2020), 1–28.

[93] Andreas Mang and George Biros. 2015. An inexact newton–krylov algorithm for constrained diffeomorphic image registration. *SIAM J. Imag. Sci.* 8, 2 (2015), 1030–1069.

[94] Giuseppe Mendicino, Jessica Pedace, and Alfonso Senatore. 2015. Stability of an overland flow scheme in the framework of a fully coupled eco-hydrological model based on the macroscopic cellular automata approach. *Commun. Nonlin. Sci. Numer. Simul.* 21, 1-3 (2015), 128–146.

[95] K. Moaddy, Shaher Momani, and I. Hashim. 2011. The non-standard finite difference scheme for linear fractional PDEs in fluid mechanics. *Comput. Math. Appl.* 61, 4 (2011), 1209–1216.

[96] Gino Moretti. 1979. The λ-scheme. *Comput. Fluids* 7, 3 (1979), 191–205.

[97] D. H. Mugler and R. A. Scott. 1988. Fast fourier transform method for partial differential equations, case study: The 2-D diffusion equation. *Comput. Math. Appl.* 16, 3 (1988), 221–228.

[98] Gerrit Mur. 1981. Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations. *IEEE Trans. Electromagn. Compat.* 4 (1981), 377–382.

[99] Cameron Musco and Christopher Musco. 2015. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Adv. Neural Inf. Process. Syst.* 28 (2015), 1396–1404.

[100] Habib N. Najm, Peter S. Wyckoff, and Omar M. Knio. 1998. A semi-implicit numerical scheme for reacting flow: I. stiff chemistry. *J. Comput. Phys.* 143, 2 (1998), 381–402.

[101] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey. 2010. 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10).* IEEE, 1–13.

[102] U. C. Nkwunonwo, Malcolm Whitworth, and Brian Baily. 2019. Urban flood modelling combining cellular automata framework with semi-implicit finite difference numerical formulation. *J. Afr. Earth Sci.* 150 (2019), 272–281.

[103] Yvan Notay and Panayot S. Vassilevski. 2008. Recursive krylov-based multigrid cycles. *Numer. Lin. Algebr. Appl.* 15, 5 (2008), 473–487.

[104] Vladimir E. Ostashev, D. Keith Wilson, Lanbo Liu, David F. Aldridge, Neill P. Symons, and David Marlin. 2005. Equations for finite-difference, time-domain simulation of sound propagation in moving inhomogeneous media and numerical implementation. *J. Acoust. Soc. Am.* 117, 2 (2005), 503–517.

[105] Tao Pang. 1999. An introduction to computational physics.

[106] A. Pereda, Luis A. Vielva, A. Vegas, and Andrés Prieto. 2001. Analyzing the stability of the FDTD technique by combining the von Neumann method with the routh-hurwitz criterion. *IEEE Trans. Microw. Theory Techn.* 49, 2 (2001), 377–381.

[107] Gabriel Peyré. 2011. The numerical tours of signal processing-advanced computational signal and image processing. *IEEE Comput. Sci. Eng.* 13, 4 (2011), 94–97.

[108] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007. *Numerical Recipes: The Art of Scientific Computing*, 3rd edition. Cambridge University Press.

[109] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *ACM SIGPLAN Not.* 48, 6 (2013), 519–530.

[110] Alfred Ramani, Basil Grammaticos, and Tassos Bountis. 1989. The painlevé property and singularity analysis of integrable and non-integrable systems. *Phys. Rep.* 180, 3 (1989), 159–245.

[111] Michel Rappaz, Michel Bellet, and Michel Deville. 2010. *Numerical Modeling in Materials Science and Engineering*. Vol. 32. Springer Science & Business Media.

[112] Prashant Singh Rawat, Miheer Vaidya, Aravind Sukumaran-Rajam, Mahesh Ravishankar, Vinod Grover, Atanas Rountev, Louis-Noël Pouchet, and P. Sadayappan. 2018. Domain-specific optimization and generation of high-performance GPU code for stencil computations. *Proc. IEEE* 106, 11 (2018), 1902–1920.

[113] Ludovic Renson, Gaëtan Kerschen, and Bruno Cochelin. 2016. Numerical computation of nonlinear normal modes in mechanical engineering. *J. Sound Vibr.* 364 (2016), 177–206.

[114] André Robert. 1981. A stable numerical integration scheme for the primitive meteorological equations. *Atmosph. Ocean* 19, 1 (1981), 35–46.

[115] Andre Robert. 1982. A semi-lagrangian and semi-implicit numerical integration scheme for the primitive meteorological equations. *J. Meteorol. Soc. Jpn. Ser. II* 60, 1 (1982), 319–325.

[116] Gian-Carlo Rota, David Kahaner, and Andrew Odlyzko. 1973. On the foundations of combinatorial theory. VIII. Finite operator calculus. *J. Math. Anal. Appl.* 42, 3 (1973), 684–760.

[117] John C. Russ, James R. Matey, A. John Mallinckrodt, and Susan McKay. 1994. The image processing handbook. *Comput. Phys.* 8, 2 (1994), 177–178.

[118] Youcef Saad. 1989. Krylov subspace methods on supercomputers. *SIAM J. Sci. Statist. Comput.* 10, 6 (1989), 1200–1232.

[119] Hasan I. Saleheen and Kwong T. Ng. 1997. New finite difference formulations for general inhomogeneous anisotropic bioelectric problems. *IEEE Trans. Biomed. Eng.* 44, 9 (1997), 800–809.

[120] Kentaro Sano, Yoshiaki Hatsuda, and Satoru Yamamoto. 2011. Scalable streaming-array of simple soft-processors for stencil computations with constant memory-bandwidth. In *Proceedings of the IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 234–241.

[121] Ulrich Schumann and Roland A. Sweet. 1988. Fast fourier transforms for direct solution of poisson's equation with staggered boundary conditions. *J. Comput. Phys.* 75, 1 (1988), 123–137.

[122] Mateusz Sitko, Maciej Pietrzyk, and Lukasz Madej. 2016. Time and length scale issues in numerical modelling of dynamic recrystallization based on the multi space cellular automata method. *J. Comput. Sci.* 16 (2016), 98–113.

[123] J. A. Somers. 1993. Direct simulation of fluid flow with cellular automata and the lattice-Boltzmann equation. *Appl. Sci. Res.* 51, 1-2 (1993), 127–133.

[124] Mario A. Storti, Rodrigo R. Paz, Lisandro D. Dalcin, Santiago D. Costarelli, and Sergio R. Idelsohn. 2013. A FFT preconditioning technique for the solution of incompressible flow on GPUs. *Comput. Fluids* 74 (2013), 44–57.

[125] John C. Strikwerda. 2004. *Finite Difference Schemes and Partial Differential Equations*. SIAM.

[126] Peter K. Sweby. 1984. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM J. Numer. Anal.* 21, 5 (1984), 995–1011.

[127] Rudolph Szilard. 2004. Theories and applications of plate analysis: Classical, numerical and engineering methods. *Appl. Mech. Rev.* 57, 6 (2004), B32–B33.

[128] Allen Taflove and Susan C. Hagness. 2005. *Computational Electrodynamics: The Finite-difference Time-domain Method*. Artech House.

[129] Christopher K. W. Tam and Zhong Dong. 1994. Wall boundary conditions for high-order finite-difference schemes in computational aeroacoustics. *Theor. Comput. Fluid Dynam.* 6, 5 (1994), 303–322.

[130] Yuan Tang, Rezaul Alam Chowdhury, Bradley C Kuszmaul, Chi-Keung Luk, and Charles E Leiserson. 2011. The pochoir stencil compiler. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*. 117–128.

[131] Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson. 2011. The pochoir stencil compiler. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures*. 117–128.

[132] Fernando L. Teixeira. 2008. Time-domain finite-difference and finite-element methods for maxwell equations in complex media. *IEEE Trans. Antennas Propagat.* 56, 8 (2008), 2150–2166.

[133] Fernando L. Teixeira, Weng Cho Chew, Mark Straka, M. L. Oristaglio, and T. Wang. 1998. Finite-difference time-domain simulation of ground penetrating radar on dispersive, inhomogeneous, and conductive soils. *IEEE Trans. Geosci. Remote Sens.* 36, 6 (1998), 1928–1937.

[134] Jos Thijssen. 2007. *Computational Physics*. Cambridge University Press.

[135] Eli Turkel. 1987. Preconditioned methods for solving the incompressible and low speed compressible equations. *J. Comput. Phys.* 72, 2 (1987), 277–298.

[136] Evgenij E. Tyrtyshnikov. 1996. A unifying approach to some old and new theorems on distribution and clustering. *Lin. Algebr. Appl.* 232 (1996), 1–43.

[137] Henk A. Van der Vorst. 2003. *Iterative Krylov Methods for Large Linear Systems*. Number 13. Cambridge University Press.

[138] Jeffrey P. Van Doormaal and George D. Raithby. 1984. Enhancements of the SIMPLE method for predicting incompressible fluid flows. *Numer. Heat Transf.* 7, 2 (1984), 147–163.

[139] Bram Van Leer. 1979. Towards the ultimate conservative difference scheme. V. A second-order sequel to godunov's method. *J. Comput. Phys.* 32, 1 (1979), 101–136.

[140] Ursula Van Rienen. 2012. *Numerical Methods in Computational Electrodynamics: Linear Systems in Practical Applications*. Vol. 12. Springer Science & Business Media.

[141] Sven Verdoolaege, Juan Carlos Juega, Albert Cohen, Jose Ignacio Gomez, Christian Tenllado, and Francky Catthoor. 2013. Polyhedral parallel code generation for CUDA. *ACM Trans. Arch. Code Optim.* 9, 4 (2013), 1–23.

[142] Luminita A. Vese and Stanley J. Osher. 2002. Numerical methods for p-harmonic flows and applications to image processing. *SIAM J. Numer. Anal.* 40, 6 (2002), 2085–2104.

[143] Franz J. Vesely. 1994. *Computational Physics*. Springer.

[144] Jean Virieux. 1986. P-SV wave propagation in heterogeneous media: Velocity-stress finite-difference method. *Geophysics* 51, 4 (1986), 889–901.

[145] Tsili Wang and Gerald W. Hohmann. 1993. A finite-difference, time-domain solution for three-dimensional electromagnetic modeling. *Geophysics* 58, 6 (1993), 797–809.

[146] Joachim Weickert. 1996. Theoretical foundations of anisotropic diffusion in image processing. In *Theoretical Foundations of Computer Vision*. Springer, 221–236.

[147] Joachim Weickert. 2000. Applications of nonlinear diffusion in image processing and computer vision.

[148] Pieter Wesseling. 1996. von neumann stability conditions for the convection-diffusion eqation. *IMA J. Numer. Anal.* 16, 4 (1996), 583–598.

[149] Shmuel Winograd. 1978. On computing the discrete fourier transform. *Math. Comput.* 32, 141 (1978), 175–199.

[150] Michael E. Wolf and Monica S. Lam. 1991. A data locality optimizing algorithm. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. 30–44.

[151] Michael E. Wolf, Dror E. Maydan, and Ding-Kai Chen. 1996. Combining loop transformations considering caches and scheduling. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 274–286.

[152] Michael J. Wolfe. 1987. Iteration space tiling for memory hierarchies. *Parallel Process. Sci. Comput.* 357 (1987), 361.

[153] David Wonnacott. 2002. Achieving scalable locality with time skewing. *Int. J. Parallel Program.* 30, 3 (2002), 181–221.

[154] Aiguo Xu, G. Gonnella, and A. Lamura. 2006. Simulations of complex fluids by mixed lattice boltzmann-finite difference methods. *Phys. A: Stat. Mech. Appl.* 362, 1 (2006), 42–47.

[155] Tomofumi Yuki and Louis-Noël Pouchet. 2015. Polybench 4.0.

[156] Santos B. Yuste and Luis Acedo. 2005. An explicit finite difference method and a new von neumann-type stability analysis for fractional diffusion equations. *SIAM J. Numer. Anal.* 42, 5 (2005), 1862–1874.

Received 16 February 2022; accepted 27 June 2023