# FlashFFTStencil: Bridging Fast Fourier Transforms to Memory-Efficient Stencil Computations on Tensor Core Units

Haozhi Han*
Peking University
Microsoft Research
Beijing, China

Kun Li†
Microsoft Research
Beijing, China

Wei Cui
Microsoft Research
Vancouver, Canada

Donglin Bai
Microsoft Research
Beijing, China

Yiwei Zhang*
University of Chinese Academy of Sciences
Microsoft Research
Beijing, China

Liang Yuan
Chinese Academy of Sciences
Beijing, China

Yifeng Chen
Peking University
Beijing, China

Yunquan Zhang
Chinese Academy of Sciences
Beijing, China

Ting Cao
Microsoft Research
Beijing, China

Mao Yang
Microsoft Research
Beijing, China

## Abstract

While Tensor Core Units (TCUs) excel in AI tasks, their application to HPC algorithms like stencil computations faces significant challenges due to sparsity, which leads to underutilization and exacerbates memory-bound limitations. This paper introduces FlashFFTStencil[1], a memory-efficient stencil computing system designed to bridge FFT to fully-dense stencil computations on TCUs. Aimed at bound shifting, FlashFFTStencil comprises three key techniques: *Kernel Tailoring on HBM* fuses distinct kernels to enhance parallelism while reducing memory transfer and footprint; *Architecture Aligning on SMEM* restructures FFT-based stencil computations into dense matrix multiplications tailored for shared memory architecture; *Computation Streamlining on TCU* optimizes TCU utilization and thread parallelism by minimizing pipeline stalls and maximizing register reuse. Notably, a distinctive extension is FlashFFTStencil's ability to enable theoretically unrestricted temporal fusion by FFT. Results show that FlashFFTStencil achieves effective sparsity-free bound shifting, with an average speedup of 2.57x over the state-of-the-art. FlashFFTStencil pioneers a new era in unifying computational patterns within the HPC landscape and bridges them with cutting-edge AI-driven hardware innovations like TCUs.

***CCS Concepts:*** • **Computing methodologies** → **Parallel algorithms**; • **Computer systems organization** → Parallel architectures.

***Keywords:*** Stencil Computation, Fast Fourier Transforms, Matrix Multiplication, Tensor Core Units

## 1 Introduction

Deep learning has ignited transformative advancements across various domains. Since most deep learning models rely on matrix multiplication (MM) operations [17], both existing and emerging AI infrastructures have increasingly incorporated specialized units known as Tensor Core Units (TCUs) to expedite these computations, with Tensor Cores in NVIDIA GPUs being a prominent example.

Currently, TCUs are not only popular in deep learning but also act as important accelerators in more HPC systems. Many world-leading supercomputers, including 8 out of the top 10 in the latest TOP500 [68] list, are equipped with TCUs to boost extra performance. Despite the prevalent use of MM in deep learning models, computational patterns in the HPC domain are becoming increasingly diverse, extending

*Work done during an internship at Microsoft Research.
†Corresponding author (kunli@microsoft.com).
[1]FlashFFTStencil is available at https://github.com/HPHEX/FlashFFTStencil.



*PPoPP '25, March 1–5, 2025, Las Vegas, NV, USA*

ACM ISBN 979-8-4007-1443-6/25/03
https://doi.org/10.1145/3710848.3710897

beyond just MM. This diversity often leads to TCUs being underutilized in practical HPC applications.

The Berkeley View identifies seven widely-used performance-critical computing patterns in the HPC field, known as *dwarfs*. Among these, stencil computation is one of the most frequently employed [6, 7, 32]. This pattern involves calculating the value of a cell in a spatial grid at a given time step based on the values of neighboring cells from previous time steps [33, 66]. It is extensively used to simulate the evolution of physical systems over time in various applications, including fluid dynamics [27, 37], electromagnetics [38], earth modeling [28], and meteorology [5, 9, 61]. Although stencil computations are crucial for scientific and engineering applications, they cannot be directly mapped to MM, which is the sole operation supported by TCUs.

Recent works, such as ConvStencil [15], LoRAStencil [74], and TCStencil [36], share the core idea of transforming the data layout of stencil computation into formats that can be reinterpreted as MM and then mapped onto TCUs. However, since the specific shape of MM on TCUs typically misaligns with diverse stencil patterns, these transformations inevitably introduce significant *sparsity*, i.e., operations on matrices with many zeros [15, 36]. These zeros are populated to mask redundant calculations for correct results, leading to a waste of TCU compute power. Moreover, given that stencil computations are inherently memory-bound, the increased sparsity intensifies this performance bottleneck by amplifying data movements between high-bandwidth memory (HBM) and registers.

To illustrate the impact of this sparsity, we utilize *arithmetic intensity*, defined as the ratio of arithmetic operations to data movement, to pinpoint the bound of current methods [11, 13, 51]. For FP64 Tensor Cores on the A100 [44], an arithmetic intensity of at least 10.1 is required to fully activate the capabilities of TCUs [1, 53]. However, due to the sparsity in the newly released LoRAStencil, which ranges from 56.3% to 71.9%, the achieved arithmetic intensity averages only 7.41 [74], falling short of the necessary threshold. ConvStencil and TCStencil exhibit even lower arithmetic intensities, at 3.59 and 2.78, respectively [15, 36]. To the best of our knowledge, no existing work effectively addresses the ever-widening gap between the stronger compute power of TCUs and the memory-bound demands of stencil computations.

This paper presents a memory-efficient stencil computing system, FlashFFTStencil, designed to bridge Fast Fourier Transforms (FFT) to fully-dense stencil computations on Tensor Core Units with far fewer memory accesses.

The design of FlashFFTStencil is grounded in three crucial observations: (1) Stencil computations in high-performance computing and convolutions in deep learning exhibit high similarities, both involving local weighted sum operations [15]; (2) According to the convolution theorem, memory-intensive

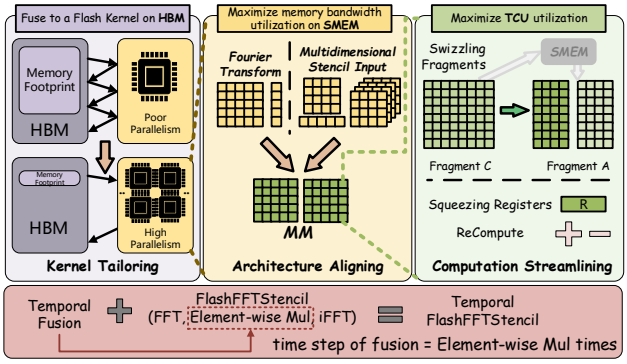


**Figure 1.** FlashFFTStencil Overview

convolution operations in the time domain can be transformed into more computationally efficient multiplication operations in the frequency domain [31, 34]; (3) FFT contributes to an efficient conversion from time-domain signals to frequency-domain signals [23]. Guided by these observations, the key insight is inspired: Given that TCUs are engineered to accelerate matrix multiplications, *can we leverage FFT as a bridge to convert the expensive memory operations into TCU-digestible matrix multiplications for bound shifting?*

Despite the shaped insights, efficiently building the FlashFFTStencil system remains a daunting challenge at both the algorithmic and hardware levels. At the algorithmic level, applying FFT to stencil computations via matrix operations to increase TCU compute workloads is far from straightforward. Although the convolution theorem applies to certain convolutions, stencil computations vary significantly in kernel shape and dimensionality. Additionally, time-domain convolutions are converted to frequency-domain multiplications through FFT, rather than directly into matrix multiplications, further complicating the design process. At the hardware level, the hierarchical memory architecture of GPUs introduces varying access latencies and capacities, with larger memory capacities typically accompanied by higher access latencies. Efficiently aligning the memory access strategies of FlashFFTStencil with these diverse memory tiers to mitigate bottlenecks necessitates meticulous designs and careful implementation. As illustrated in Figure 1, FlashFFTStencil comprises three key techniques to address these challenges:

(1) Kernel Tailoring on HBM. Kernel Tailoring fuses distinct kernels into a single, optimized kernel. It splits the original input and distributes these segments across different thread blocks, enabling the GPU to execute the fused FFT-based stencil computations within on-chip memory with a high degree of parallelism. By Kernel Tailoring, FlashFFTStencil achieves over a 3x memory transfer reduction to and from HBM and an average 9x reduction in memory footprint.

(2) Architecture Aligning on SMEM. First, 2D Dimension Alignment is presented to reconfigure the multidimensional

FFT-based stencil computation into 2D matrix multiplications, converting sparse matrices (up to 87.5% sparsity) into fully-dense matrices (0% sparsity).Then Diagonal Data Indexing is designed for fully coalesced global memory access while ensuring shared memory access entirely free from bank conflicts. Additionally, Double-layer Filling repurposes the complex number format inherent in FFT for real-number stencil computations, enabling full reuse of both computation and storage through complex number operations.

(3) Computation Streamlining on TCU. By employing Swizzling Fragments, FlashFFTStencil eliminates pipeline bubbles caused by frequent communication with shared memory, increasing TCU utilization from 54% to 76%. Additionally, it maximizes thread parallelism through Squeezing Registers, a technique that analyzes the lifecycle of registers to retain those about to be reused, thereby freeing up additional registers and doubling the number of active threads.

The properties of FFT are further leveraged to enhance temporal fusion, presenting a surprising advantage of FlashFFTStencil. Current work requires pre-calculating fusion weights for each dimension, leading to parameter explosion and limiting fusion to a few steps (e.g., ConvStencil to 3, LoRAStencil to 3). Multiplying the Fourier-transformed input by the Fourier-transformed stencil raised to the power of time steps theoretically enables unrestricted temporal fusion.

We conduct a comprehensive evaluation of FlashFFTStencil across various configurations on A100 and H100 GPUs. The results demonstrate that FlashFFTStencil achieves exceptionally high arithmetic intensity, fully utilizing TCUs at 100% capacity, and delivers an average speedup of up to 2.57x compared to the state-of-the-art.

***Takeaway*** Aimed at bound shifting, FlashFFTStencil effectively bridges the widening gap between the memory-intensive demands of stencil computation and the stronger computing power of TCUs, extending the boundaries of TCU in HPC applications. FlashFFTStencil pioneers a new era in the unification of computational patterns within the HPC landscapes—integrating FFT, MM, and stencil computations—and catalyzes the convergence of traditional HPC methodologies with cutting-edge AI-driven hardware innovations like TCU.

## 2 Background and Motivation

### 2.1 Memory Hierarchy and Tensor Cores on GPU

The NVIDIA GPU architecture is structured around a scalable array of multithreaded Streaming Multiprocessors (SMs), optimized for the efficient execution of highly parallel tasks [45, 46]. The fundamental execution unit within an SM is a warp, comprising 32 threads, each with its own dedicated set of registers [50]. Threads within the same block share access to a programmable shared memory space, while all threads can access global memory. As shown in Table 1, the memory hierarchy adheres to the principle that larger memory capacities correspond to higher access latencies [44, 53].

Tensor Cores in SMs are specialized hardware units engineered to expedite matrix multiplication and accumulation (MMA), as delineated in Equation (1). These cores offer significantly higher performance than traditional CUDA cores for MMA tasks [65].

$$D_{m\times n} = A_{m\times k} \times B_{k\times n} + C_{m\times n} \tag{1}$$

Tensor Cores can be programmed at the warp level using mma instructions in PTX or WMMA API [50]. The WMMA API uses a specialized data structure called fragments pfor MMA operations. Developers load data tiles from global or shared memory into these fragments, which are distributed across the registers of a warp in a specific configuration (e.g., current support in FP64 precision accommodating only with $m = 8$, $n = 8$ and $k = 4$ in Equation (1)) [45, 46].

### 2.2 Stencil Computation

Stencil computation, a widely used algorithm in scientific and engineering fields, performs a sequence of temporal sweeps (called time steps) to iteratively update the value of each point in a $d$-dimensional spatial grid based on a specific computation pattern [66]. The value of a point at time $t$ is a weighted sum of the values of its neighboring points from the previous time steps. Stencil computations in high-performance computing and convolutions in deep learning demonstrate analogous computational patterns, as both involve using a kernel to perform weighted computations on a sliding window over an input matrix.

### 2.3 FFT-based Convolution

According to the Convolution Theorem, FFT offers a more efficient computing method for convolutions by transforming operations to the Fourier domain [54]. This theorem states that the convolution of two functions in the time domain, $Y(t) = X(t) * K(t)$, is equivalent to the element-wise multiplication of their Fourier transforms in the frequency domain, followed by an inverse Fourier transform to return to the time domain:

$$Y(t) = \mathcal{F}^{-1}\{\mathcal{F}\{X(t)\} \cdot \mathcal{F}\{K(t)\}\}.$$

By applying FFT, the convolution operation can be transformed from irregular local point-by-point accumulation operations into efficient frequency-domain multiplication operations. This conversion significantly enhances both memory

**Table 1.** Memory Hierarchy

| Memory Types | Memory Capacity | Latency (cycles) |
|---|---|---|
| Global Memory | 80 GiB / GPU | 290 |
| Max Shared Memory | 164 KiB / SM | 22 |
| Max 32-bit Registers | 64 Ki / SM | 1 |

1: $k_f \leftarrow \mathcal{F}(k, N)$
2: **for** $t \leftarrow 1$ to $T$ **do**
3: $\quad a_t \leftarrow \mathcal{F}(a_{t-1}, N)$
4: $\quad a_t \leftarrow a_t * k_f$
5: $\quad a_t \leftarrow \mathcal{F}^{-1}(a_t, N)$
6: **end for**
7: **return** $a_T$

1: **for** $c \leftarrow 0$ to $Channel - 1$ **do**
2: $\quad k_f \leftarrow \mathcal{F}(k, N)$
3: $\quad a_1 \leftarrow \mathcal{F}(a_0, N)$
4: $\quad a_1 \leftarrow a_0 * k_f$
5: $\quad a_{out} \leftarrow \mathcal{F}^{-1}(a_1, N)$
6: **end for**
7: **return** $a_{out}$ in all channel

**Figure 2.** Comparison of FFT-based Stencil (Left) and Convolution (Right)

access patterns and computational efficiency, and has been widely adopted in numerous deep learning applications.

### 2.4 Opportunity: Sparks from the Collision of Stencil, Convolution, FFT, MM, and TCU

Applying FFT to stencil computations, similar to its use in convolutions, transforms point-by-point local accumulation operations into simpler multiplications in the frequency domain. These multiplications, being compute-intensive rather than memory-bound, help alleviate the memory limitations typically associated with stencil computations.

In this way, if all computations in stencil computations can be transformed into compute-intensive multiplications and then carefully optimized into matrix multiplications, thereby significantly alleviating the memory bottleneck in stencil computations, TCUs will substantially accelerate the entire computational process due to their higher peak computational performance.

### 2.5 The Elusive Oasis: Why Existing Work Can't Quench the Thirst with This Opportunity?

Although the opportunity appears highly promising, significant challenges impede the practical realization of this insight when striving for a holistic design on HBM, SMEM, and TCUs. As shown in Figure 2, unlike **convolutions**, the substantially larger input sizes and the constraint of a single channel in **stencil** computations obstruct any parallelism that relies on large channels, leading early works (e.g. cuDNN/cuFFT) to suffer from limited parallelism and poor HBM utilization [16, 30, 47, 48, 64] (Section 3.1). Furthermore, the dimensions and computational patterns of **FFT-based stencil** and **MM** introduce diverse alignments on GPU Shared Memory and Registers, which is not addressed by many works designed only for special dimensions [19, 34, 54] (Section 3.2). Even more critically, **TCU**-related works (e.g. LoRAStencil/ConvStencil/TCStencil) remain significantly constrained by factors such as pipeline efficiency and register occupancy [15, 36, 74] (Section 3.3). Additionally, the absence of temporal iterations in convolutions—an intrinsic feature of stencil computations—presents a substantial opportunity to integrate time-step iteration optimizations into FFT-based stencil approaches on GPU, a distinctive feature that has not yet been explored by other studies [2–4, 35] (Section 4).

The complexity arising from the interplay of Stencil, Convolution, FFT, MM, and TCU creates a web of genuine challenges, particularly when interwoven with diverse characteristics of memory architectures. To the best of our knowledge, this intricate network of challenges has not been effectively discerned or addressed by existing research.

## 3 FlashFFTStencil: System Design

### 3.1 Kernel Tailoring on HBM

The standard FFT stencil computation is significantly constrained by extensive I/O operations with HBM. These constraints are primarily due to the substantial memory transfers required during the computation process. Firstly, computing the stencil results for a single timestep involves processing the original input data through three distinct kernels: FFT, element-wise multiplication, and iFFT. Each of these kernels requires reading data from and writing data to HBM, necessitating repetitive data round-trips between SRAM and HBM. Secondly, these three kernels also depend on substantial auxiliary data stored in HBM, which must be loaded into SRAM during computation. As the size of the input data increases, the amount of auxiliary data grows quadratically, further exacerbating memory bandwidth consumption and introducing latency that hinders overall computational efficiency.

The most common approach to accelerate I/O-bound computations is kernel fusion: when multiple operations are sequentially applied to the same input, kernel fusion allows the input to be loaded only once, rather than multiple times by different kernels. However, the limited capacity of on-chip memory makes it challenging to accommodate the entire input and perform the computations of all three operators on SRAM. Moreover, in the context of FFT stencil computations, poor parallelism is constrained by global data dependencies. This makes it difficult to distribute data across the SRAM of each Streaming Multiprocessor (SM) and achieve effective parallel computation through tiling.

Here, Kernel Tailoring is presented to achieve efficient memory management and enhanced parallelism at the HBM level. It consists of three key steps: Splitting the original input into smaller segments that can be accommodated in SRAM for parallel computing; Each segment undergoes a fusing operation that includes FFT, element-wise multiplication, and iFFT; The outputs, distributed across different SMs, are gathered back to HBM through a stitching process.

**Splitting.** Given the input $\mathbf{X} \in \mathbb{R}^N$ and kernel $\mathbf{K} \in \mathbb{R}^M$, where $N$ represents the length of the original input data and $M$ indicates the length of the stencil computation kernel, the stencil result for a single timestep can be calculated using Equation (2), based on the definition of stencil computations

and the circular convolution theorem [31].

$$\mathrm{Y}[n] = (\mathrm{X} * \mathrm{K})\,[n] \triangleq \sum_{m=0}^{M-1} \mathrm{K}[m] \cdot \mathrm{X}[n-m] \tag{2}$$

Kernel Tailoring splits the original input into multiple segments and loads them from slow HBM to fast SRAM. Each segment is denoted as $\mathbf{x} \in \mathbb{R}^L$, where $L$ represents the length of each segment after splitting. The process of splitting can be expressed by Equation (3).

$$\mathbf{x}[l] \triangleq \begin{cases} \mathbf{X}[l+iS], & 0 \le l \le L-1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

$\mathbf{x}_i$ represents the data within the $i$-th segment after splitting. $S$ denotes the size of the stencil result derived from the current segment. This size is determined by the segment length $L$, as expressed in Equation (4).

$$S \le L - (M-1) \tag{4}$$

The value of $L$ is elastically adjustable and can be auto-tuned to the most efficient value based on on-chip memory of capacity $C$ and the matrix fragment size on the TCU $T$, as shown in Equation (5).

$$L = aT(T-1), 2aT^2 \cdot p \le C, a \in \mathbb{Z}^+ \tag{5}$$

$p$ denotes the number of blocks executed simultaneously on a single Streaming Multiprocessor (SM).

As shown in Step 1 of Figure 3, the original input is split into segments, which are then loaded into SRAM on different SMs. Each segment undergoes parallel stencil computation across different blocks in SMs, producing the results $\mathbf{y}_i$ for that segment, as shown in Equation (6).

$$\mathbf{y}_i[l] = (\mathbf{x}_i * \mathbf{K})\,[l] \triangleq \sum_{m=0}^{M-1} \mathbf{K}[m] \cdot \mathbf{x}_i[l-m] \tag{6}$$

For $n \in [iS+M, iS+S+M-1]$, $\mathbf{X}[n]$ and $\mathbf{x}_i[l]$ have the same values, where $l$ belongs to $[M, S+M-1]$. Therefore, as shown in Equation (7), by computing the stencil result for the segment in SRAM, the corresponding stencil result for the original input can be obtained.

$$\mathrm{Y}[n] \triangleq \sum_{m=0}^{M-1} \mathrm{K}[m] \cdot \mathrm{X}[n-m] = \sum_{m=0}^{M-1} \mathbf{K}[m] \cdot \mathbf{x}_i[l-m] \triangleq \mathbf{y}_i[l] \tag{7}$$

**Fusing.** For each partitioned segment of data, stencil computations are performed in SRAM. As illustrated in Figure 3, the original input data of length $N$ is partitioned into multiple segments, each of length $L$. Within each segment, the data dependencies of the FFT operations are restricted to that segment, allowing FFT stencil computations to be executed in parallel across different segments. Consequently, these segments are assigned to different SMs for parallel computation.

Each segment sequentially loads the FFT matrix $\mathbf{F}$, the Fourier-transformed kernel data $\mathbf{k_f}$, and the iFFT matrix $\mathbf{iF}$ from HBM to perform FFT, element-wise multiplication, and iFFT operations. Upon completion of each operation, the data does not need to be written back to HBM and subsequently read again. Instead, the next operation is executed directly in SRAM. This approach effectively fuses the execution of different kernels within SRAM.

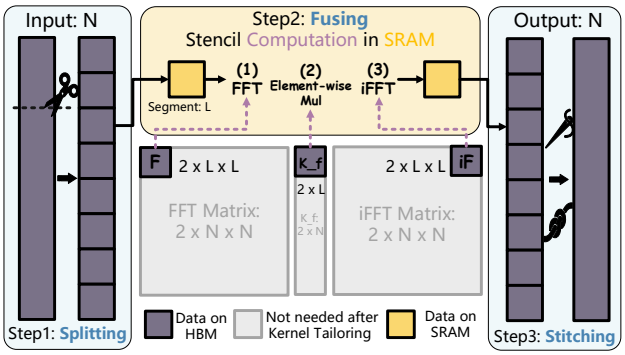


**Figure 3.** Kernel Tailoring of FlashFFTStencil

For an original data size of $N$, the standard FFT stencil computation requires auxiliary data totaling $2(2N^2+N)$, which includes the FFT/iFFT matrices and $\mathbf{k_f}$, to be stored in HBM. After splitting, each segment shares the same auxiliary data, with only one set of auxiliary data, of size $2(2L^2+L)$, where $L \ll N$, needing to be stored in HBM. As illustrated in Figure 3, the grey area represents the memory savings achieved through Kernel Tailoring, which significantly alleviates the memory capacity requirements for FFT stencil computations on GPUs.

**Stitching.** By performing stencil computation on each segment $\mathbf{x}_i$, we can calculate the stencil result $\mathbf{y}_i$. As demonstrated in Equation (7), the computed result $\mathbf{y}_n$ for each block is equivalent to the output $\mathbf{y}$. The subregion $[M, S+M-1]$ in $\mathbf{y}_i$ is then stitched into the overall output data. Through this stitching process, the stencil results for the entire input are accurately computed, mathematically equivalent to those obtained from the standard FFT stencil computation.

By executing the aforementioned three steps—splitting, fusing, and stitching—Kernel Tailoring significantly reduces memory transfers, lowers auxiliary data requirements, and enhances parallelism efficiency at the HBM level.

## 3.2 Architecture Aligning on SMEM

**3.2.1 2D Dimension Alignment.** The fragment structure on TCUs is inherently two-dimensional, aligning with MMA computations. In contrast, stencil computations are multidimensional, leading to a *dimensional conflict* when transferring data from SMEM to TCU. This conflict is particularly pronounced in one-dimensional stencil computations, where the Fourier transform involves matrix-vector multiplications. However, TCUs, optimized for matrix-matrix multiplications, are inefficient when executing matrix-vector multiplications. For example, at FP64 precision, Tensor Cores on the Nvidia A100 only support 8 × 4 × 8 MMA operations [65], leaving

7 out of 8 columns in the right-hand matrix underutilized. This underutilization results in computational sparsity, significantly diminishing TCU efficiency. To address this, we propose three steps to align multidimensional input data into a logically two-dimensional format, enhancing the efficiency of TCUs in accelerating stencil computation.

**Prime-Factor FFT Algorithm (PFA) for Reshaping.** PFA aims to reshape a one-dimensional Fourier transform of size $N$ into a two-dimensional Fourier transform of size $N_1 \times N_2$, where $N_1$ and $N_2$ are co-prime integers [12, 20]. By decomposing the computation into smaller, manageable parts, PFA aligns one-dimensional data into a logically two-dimensional format on SMEM, enabling the following efficient matrix-matrix multiplications on TCUs.

**Chinese Remainder Theorem (CRT) for Reordering in PFA.** CRT provides a way to solve systems of simultaneous congruences with pairwise co-prime moduli [52]. It ensures that a unique solution exists and can be efficiently computed, which is critical for reordering data in PFA. Let $N = N_1N_2$, where $N_1$ and $N_2$ are relatively prime. The index $n$ of the one-dimensional data and the row/column index $n_1/n_2$ of the reshaped two-dimensional data format satisfy the relationship given in Equation (8).

$$n = n_1N_2 + n_2N_1 \mod N \tag{8}$$

Given that $N_1$ and $N_2$ are relatively prime, CRT ensures that for each $n$, there exist integers $n_1$ and $n_2$ satisfying the above equation. Furthermore, modulo $N$, $n_1$ can be adjusted to lie between 0 and $N_1 - 1$, and $n_2$ can be adjusted to lie between 0 and $N_2 - 1$.

**Multidimensional Data Handling.** Higher-dimensional data (e.g., 3D or more) can also be processed in 2D slices, as shown in Figure 4. By decomposing multidimensional data into 2D slices, the data can be efficiently adapted for TCU operations without encountering computational sparsity. Additionally, this approach facilitates the distribution of data across multiple SMs, enabling parallel processing, with each SM independently handling a specific slice of the data.

### 3.2.2 Diagonal Data Indexing.

FlashFFTStencil achieves the mapping of multidimensional data to TCUs for computation. However, the data reordering process of PFA on GPUs is highly time-consuming. This process involves numerous modulo operations, which are computationally inefficient on GPUs. Furthermore, data reordering involves random memory accesses, which can easily lead to bank conflicts in SMEM, thereby reducing memory bandwidth utilization.

In this section, we address the aforementioned problem with a novel diagonal indexing strategy that employs fine-grained control of bank access, thereby eliminating the need for data reordering in the PFA algorithm. Our approach is founded on the following observations: *Observation 1:* The bank mechanism in SMEM guarantees that memory access efficiency is decoupled from the continuity of memory access addresses. *Observation 2:* The data reordering process in PFA can be supplanted by a specialized data indexing approach. *Observation 3:* The data indexing process can obviate the time-intensive modulo operations involved in the reordering method of PFA.

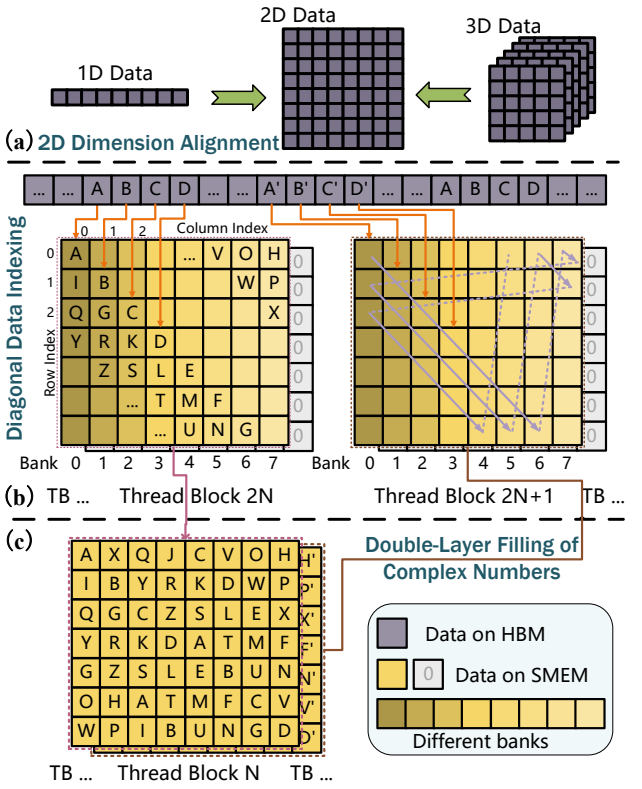


**Figure 4.** Architecture Aligning of FlashFFTStencil, (a) illustrates 2D Dimension Alignment, (b) depicts Diagonal Data Indexing assuming the bank length is eight, and (c) represents Double-layer Filling of Complex Numbers.

After employing the Kernel Tailoring technique, multiple segments are managed by distinct thread blocks on the SM. As depicted in Figure 4(b), the threads within each thread block coalesce to read the data of the current segment from HBM and store it to SMEM. The thread block employs coalesced memory transactions as the fundamental unit for accessing HBM, thereby optimizing the utilization of its memory bandwidth. Threads within each thread block store data into SMEM using the diagonal data indexing strategy where both the row index $n_1$ and the column index $n_2$ are incremented continuously. For instance, as illustrated in Figure 4(b), the first data element in the current segment, A, will be stored at the position with the row/column index 0/0 on SMEM. The row/column index of the second data element B will increment to 1/1. Following the diagonal indexing strategy, elements C and D will be stored at the position with the row/column index of 2/2 and 3/3. When either the row

or column index exceeds $N_1$ or $N_2$, respectively, the index resets to 0 and continues incrementing until all data from the segment has been loaded into SMEM. On the right side of Figure 4(b), the purple lines within the SMEM of thread block 2N+1 indicate the trace of data storage in SMEM. This ensures that during each memory transaction, all threads within a thread block access different banks in SMEM. This approach maximizes the utilization of memory bandwidth from HBM to SMEM.

By fully leveraging memory optimization techniques of the GPU architecture, the additional challenges introduced by Dimension Alignment can be effectively addressed. This enables efficient Dimension Alignment from HBM to SMEM for FlashFFTStencil.

#### 3.2.3 Double-layer Filling of Complex Numbers.

Following Dimension Alignment and Diagonal Indexing, the data is prepared on SMEM and subsequently loaded into fragments for the TCUs to perform stencil computations, comprising FFT, element-wise multiplication, and iFFT. However, there are distinct misalignments of the data types in FFT and stencil computation which we refer to as the Complex Numbers Disaster. Specifically, (1) the input and output of stencil calculations are real numbers; however, the intermediate variables in Fused Stencil Computations are complex, requiring twice the storage space of the input data; (2) complex multiplication requires four real multiplications and three real additions, significantly increasing the workload on computational units of GPUs, which are primarily designed for real number calculations. To address Complex Numbers Disaster, the computational complexity and storage overhead introduced by FFT, we propose the Double-layer Filling technique, leveraging the properties of the real number Fourier transform.

Double-layer filling leverages the conjugate symmetry of the Fourier transform of real inputs, as shown in Equation (9) [42, 62].

$$X_{n-i} = X_i^* \tag{9}$$

$X$ denotes the data obtained by performing the Fourier transform on the real input, and $*$ denotes the conjugate symmetry operation. As shown in Figure 4(c), we use another segment in next thread block as the imaginary part to fill the current segment, forming a single segment of data treated as complex data for FFT stencil computations. By employing Double-layer Filling, we can derive the stencil results for two real segments using a single FFT stencil computation. Respectively, the real and imaginary parts of the complex result correspond to the first and second segments of the stencil result.

This method of filling input data for combined computation significantly mitigates the computational explosion caused by the Complex Numbers Disaster. Additionally, it reduces the memory required to store intermediate data by half, making it equivalent to the memory footprint of the input data.

### 3.3 Computation Streamlining on TCU

Kernel Tailoring on HBM allows the FFT stencil algorithm to be executed on GPUs in a memory-efficient manner using a fused kernel. Architecture Aligning on SMEM leverages the memory optimization technology of GPUs to efficiently prepare data for optimal computation on TCUs. In this section, we introduce the Computation Streamlining of FlashFFTStencil, which is designed to achieve efficient FFT stencil computation through a series of matrix operations on TCUs.

**Algorithm 1** Computation Streamlining of FlashFFTStencil with Matrix Operations ($\otimes$ denotes matrix multiplication on TCU)

**Ensure:** $\mathbf{y_i}$ in SMEM
**Require:** FFT matrices $\mathbf{F_1}, \mathbf{F_2} \in \mathbb{C}^{N\times N}$, stencil kernel $\mathbf{k_f} \in \mathbb{C}^{N\times N}$, input data $\mathbf{x_i} \in \mathbb{C}^{N\times N}$ in SMEM
1: $\mathbf{x_i} \leftarrow (\mathbf{F_1} \otimes \mathbf{x_i}) \otimes \mathbf{F_2}$
2: $\mathbf{x_i} \leftarrow \mathbf{x_i} * \mathbf{k_f}$
3: $\mathbf{F_1^{-1}}, \mathbf{F_2^{-1}} \leftarrow \mathbf{F_1}, \mathbf{F_2}$
4: $\mathbf{y_i} \leftarrow \mathbf{F_1^{-1}} \otimes (\mathbf{x_i} \otimes \mathbf{F_2^{-1}})$
5: **return** $\mathbf{y_i}$

We present the pseudocode for Computation Streamlining of FlashFFTStencil on TCU in Algorithm 1. Once the input data has undergone 2D Dimension Alignment and is well-prepared in SMEM, it can be directly loaded into the fragments used for TCU computation. As shown in line 1, the input data, already reshaped as a matrix, undergoes the Fourier transform by being left-multiplied and right-multiplied by the FFT matrix. In line 2, the transformed $\mathbf{x_i}$ is element-wise multiplied with the Fourier-transformed stencil kernel $\mathbf{k_f}$. Line 3 illustrates the computation of the inverse FFT (iFFT) matrix using the FFT matrix. The real part of the FFT matrix is identical to that of the iFFT matrix and their imaginary parts are negatives of each other. Due to the numerical relationship between the iFFT matrix and the FFT matrix, the iFFT matrix can be easily derived from the FFT matrix. In line 4, $\mathbf{x_i}$ undergoes the inverse Fourier transform through two matrix multiplications (similar to line 1). By transforming the FFT stencil computations into matrix multiplication operations, we can significantly enhance computation performance, fully leveraging the computational capabilities of TCU.

Although the aforementioned Computation Streamline is highly efficient, profiling the initial implementation of the FlashFFTStencil algorithm revealed two significant performance bottlenecks: low TCU pipeline utilization and low parallelism within the SMs. Subsequently, we will discuss how we addressed these issues through Swizzling Fragments and Squeezing Registers.

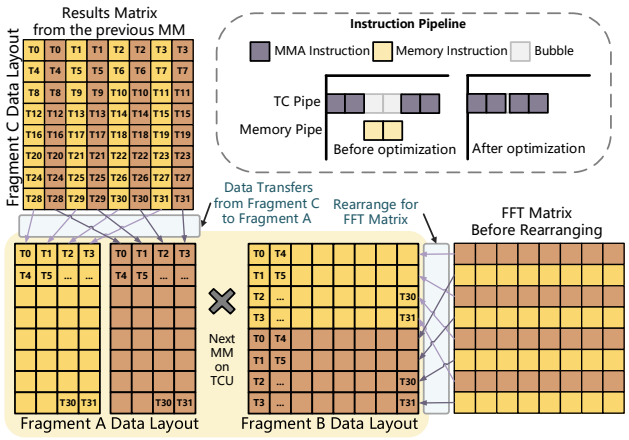


**Figure 5.** Pipeline Bubbles Removal by Swizzling Fragments

**Pipeline Bubbles Removal by Swizzling Fragments.** We identified that the low TCU pipeline utilization is attributable to the time-consuming data transfers between consecutive matrix multiplications (as shown in line 1 and line 4 of Algorithm 1). For instance, transferring data from fragment C to fragment A at FP64 precision requires SMEM for communication. This results in redundant data transfers from registers (fragment C) to SMEM and then back to registers (fragment A), causing suboptimal TCU pipeline utilization.

To address this issue, we propose Swizzling Fragments, which avoids data transfers through SMEM by directly transferring data from one register to another. Swizzling Fragments is designed based on the principle of row and column permutation invariance in matrix multiplication. As depicted in Figure 5, the result matrix from the previous matrix multiplication, stored in fragment C, can be directly transferred to fragment A without passing through SMEM by leveraging the correspondence between the data layouts of fragment C and A. Consequently, it is necessary to rearrange the rows of the FFT Matrix accordingly. This rearrangement is performed during the generation of the FFT Matrix, thereby incurring no additional storage or data transfer overhead. As shown in Figure 5, the data transfers from fragment C to fragment A correspond precisely to the rearrangement order of the FFT Matrix. This method ensures the correctness of the computation while avoiding data transfers through SMEM.

As illustrated in the instruction pipeline of Figure 5, Swizzling Fragments significantly reduces the overhead of data transfers between consecutive matrix multiplications, thereby markedly enhancing TCU pipeline utilization.

**Thread Parallelism Enhancement by Squeezing Registers.** We identified that the primary factor affecting the parallelism is the substantial demand for registers following Kernel Tailoring. For GPU kernels, excessive register usage per thread significantly impairs parallelism. To mitigate this issue, we employed Squeezing Registers and recomputation to minimize register usage, thereby maximizing parallelism.

Squeezing Registers involves utilizing the same registers to store different data at various stages of kernel execution, thereby performing distinct computational operations. For example, the fragment C involved in matrix multiplication in line 1 will be repurposed to store $\mathbf{k_f}$ and participate in element-wise multiplication in line 2. Additionally, by leveraging the numerical relationship between the FFT matrix and the iFFT matrix—where the real parts are identical and the imaginary parts are negatives of each other—we can recompute iFFT matrix from FFT matrix. This method reduces the number of registers required to store iFFT matrix, thereby minimizing overall register usage. Consequently, we employed both squeezing registers and recomputation to optimize computations, thereby enhancing thread parallelism.

Through the two optimization techniques mentioned above, we have significantly enhanced the performance of FlashFFTStencil on GPUs. These techniques enable memory-efficient stencil computations to leverage TCUs for higher computational performance.

## 4 Extension: Temporal FlashFFTStencil

Extensive research has concentrated on temporal fusion to enhance parallelism and data locality in stencil computations, thereby optimizing their overall performance. However, these methods generally rely on fixed temporal fusion, lacking flexibility. (1) This limitation is mainly due to the parameter explosion resulting from temporal fusion, which restricts fusing a limited number of timesteps in order to achieve performance enhancements. (2) Prior work on TCUs, such as ConvStencil, has been limited by the size of the TCU fragments, significantly impacting the flexibility of temporal fusion.

However, the flexibility of temporal fusion is essential for the effective application of stencil computations. In certain scenarios, the requirement to output results at each time step precludes the optimization of temporal fusion. Furthermore, In certain scenarios, application scenarios may necessitate retrieving stencil computation results after specific time steps. Thus, the flexibility of temporal fusion is one of the most critical aspects of the extensibility of stencil algorithms.

The temporal fusion of FlashFFTStencil possesses the aforementioned extensibility. The properties of the FFT Stencil algorithm are further leveraged to enhance temporal fusion, presenting a surprising advantage of FlashFFTStencil [2, 4]. It can perform element-wise multiplication on $k_f$ (i.e., $k_f * k_f$) to achieve temporal fusion, which is highly conducive to the GPU architecture. As shown in Equation (10), performing element-wise multiplication $T$ times corresponds to fusing $T$ timesteps, unimpeded by hardware or algorithmic

constraints [3, 35].

$$a_T = \mathcal{F}^{-1}(\mathcal{F}(a_0) * \underbrace{k_f * ... * k_f}_{T}) \tag{10}$$

$a_0$ denote the initial input for the stencil computation and $a_T$ denote the result after $T$ time iterations. This seamless integration of temporal fusion into FlashFFTStencil further enhances the advantages of the FlashFFTStencil algorithm.

FlashFFTStencil exhibits a performance advantage over other stencil algorithms even without temporal fusion. By incorporating temporal fusion, FlashFFTStencil not only enhances its extensibility but also further amplifies its performance superiority compared to other stencil algorithms.

# 5 Evaluation

## 5.1 Experimental Setup

**Machines.** Our experiments, conducted across distinct GPUs as detailed in Table 2, underscore the advantages of FlashFFTStencil across various hardware configurations.

**State-of-the-arts.** We conducted a comprehensive analysis by comparing FlashFFTStencil with various state-of-the-art, including cuFFT-based Stencil [30, 48, 64], cuDNN-based Stencil [16, 47], DRStencil [75–77], Brick [75–77], TCStencil [15, 36], ConvStencil [15], and LoRAStencil [74].

**Benchmarks.** To thoroughly evaluate the performance of FlashFFTStencil across diverse stencil computations, we employ stencil kernels with varying configurations, including 1D, 2D, and 3D computations. The specific details are outlined in Table 3 [15, 25, 71].

**Metrics.** We utilize both the total execution time $t$ and the $GStencil/s$ value [14, 15, 41, 70–74], which reflects the computational speed of the stencil algorithm, as key performance evaluation metrics.

## 5.2 Ablation Study

**Performance Breakdown.** In this section, we explore how FlashFFTStencil achieves performance improvements. Figure 7 presents a detailed performance breakdown of FlashFFTStencil with Heat-1D stencil kernel. Similar results are observed for other shapes of FlashFFTStencil computations, regardless of the number of time steps fused. Compared to standard FFT stencil computations (based on cuFFT best implementation), the application of Kernel Tailoring results in a 4.68x performance improvement. The introduction of FP64 Tensor Cores further enhances efficiency by 1.62x. Similarly, the Architecture Aligning and Computation Streamlining techniques provide performance gains of 1.4x and 1.08x, respectively. As illustrated in Figure 7, FlashFFTStencil achieves nearly an 11.25x speedup compared to standard FFT stencil computations. Consequently, it is evident that all the optimization techniques in FlashFFTStencil significantly contribute to performance enhancement.

**Technique I: Kernel Tailoring.** We conducted a detailed analysis of the advantages provided by Kernel Tailoring. We compared the memory footprint of FlashFFTStencil, which employs Kernel Tailoring techniques, with the best-performing standard FFT Stencil implementation based on cuFFT. As illustrated in Figure 8, FlashFFTStencil achieved a reduction in memory footprint by a factor of 7-9 compared to the best cuFFT-fft implementation.

**Technique II: Architecture Aligning.** We utilized Nsight Compute to perform a comprehensive memory workload analysis, aimed at validating the efficacy of Architecture Aligning. Table 4 presents the proportion of uncoalesced memory accesses and the average shared store bank conflicts per request for FlashFFTStencil across various stencil kernel configurations. Applying Architecture Aligning, both metrics exhibit a significant reduction.

**Technique III: Computation Streamlining.** Similarly, we conducted an analysis of the compute workload of FlashFFTStencil using Nsight Compute. Computation Streamlining enhances the efficiency of TCU utilization in FlashFFTStencil by reducing pipeline bubbles. Table 4 presents the improvements in TCU pipeline utilization.

**Table 2.** Configuration for Hardware Platforms.

| ID | GPU | FP64 | FP64 TC. | Bandwidth |
|---|---|---|---|---|
| A | NVIDIA H100 SXM 80GB | 34 TFLOPS | 67 TFLOPS | 3350 GB/s |
| B | NVIDIA A100 PCIe 80GB | 9.7 TFLOPS | 19.5 TFLOPS | 1935 GB/s |

**Table 3.** Configuration for Stencil Benchmarks

| Kernel | Kernel Points | Problem Size | Time Step |
|---|---|---|---|
| Heat-1D | 3 | $512M$ | 1000 |
| 1D5P | 5 | $512M$ | 1000 |
| 1D7P | 7 | $512M$ | 1000 |
| Heat-2D | 5 | $16K \times 16K$ | 1000 |
| Box-2D9P | 9 | $16K \times 16K$ | 1000 |
| Heat-3D | 7 | $768 \times 768 \times 768$ | 1000 |
| Box-3D27P | 27 | $768 \times 768 \times 768$ | 1000 |

**Table 4.** Memory workload analysis with or without architecture aligning and Compute workload analysis with or without computation streamlining for FlashFFTStencil with different kernels

| Metrics | Stencil Kernels | | | Avg. |
|---|---|---|---|---|
| | 1D3P | 2D9P | 3D27P | |
| UGA-w/o | 36.12% | 25.37% | 15.48% | 25.65% |
| UGA-w | 1.34% | 5.41% | 5.68% | 4.14% |
| BC/R-w/o | 1.31 | 0.97 | 0.84 | 1.04 |
| BC/R-w | 0.21 | 0.59 | 0.30 | 0.36 |
| PU-w/o | 64.32% | 59.24% | 40.06% | 54.54% |
| PU-w | 80.21% | 79.30% | 68.86% | 76.12% |

[1] UGA denotes the percentage of uncoalesced global accesses.
[2] BC/R denotes the average shared store bank conflicts per request.
[3] PU denotes the percentage of TCU pipeline Utilization.

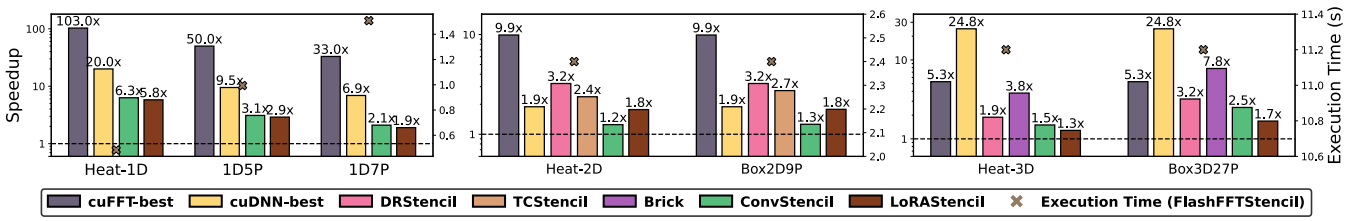


**Figure 6.** Speedup and execution time of FlashFFTStencil over the state-of-the-art stencil implementations on H100

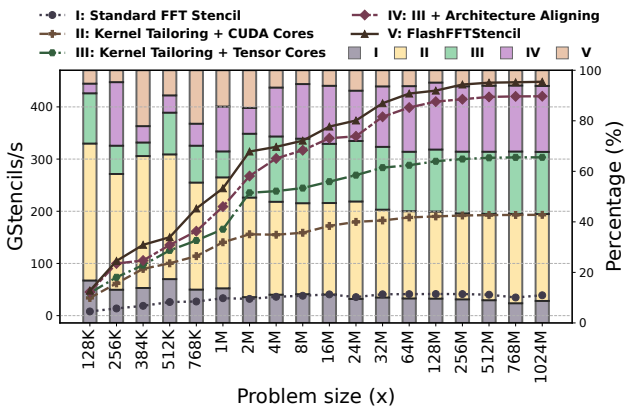


**Figure 7.** Performance breakdown of FlashFFTStencil on A100 (Heat-1D, with six time-step fusion)

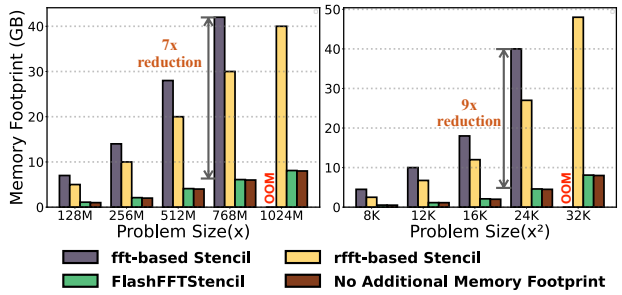


**Figure 8.** Comparison of Memory Footprint with Standard FFT Stencil in Heat-1D (left) and Box2D9P (right)

**Extension: Temporal FlashFFTStencil.** Among the state-of-the-art methods specifically developed for stencil computations, none offer the flexibility to freely control the number of fused time steps, except for cuFFT-based stencil, which are based on the theoretical foundations of the Standard FFT stencil algorithm [15, 26, 57, 74]. Consequently, as depicted in Figure 9, we compared FlashFFTStencil with the cuFFT-based stencil to validate the performance advantages of FlashFFTStencil with temporal fusion on distinct GPU.

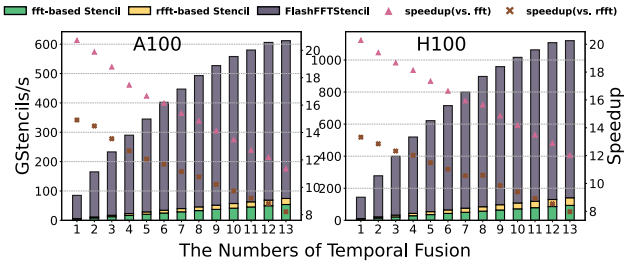


**Figure 9.** The Performance Advantages of 1D Temporal FlashFFTStencil on A100 and H100

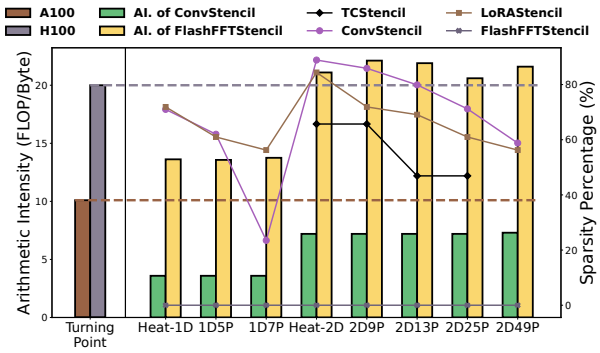


**Figure 10.** Comparison of arithmetic intensity and sparsity with TCU-based stencil computations

### 5.3 Overall Performance Comparison

We evaluate the execution time and corresponding speedup of FlashFFTStencil compared with all state-of-the-art methods on H100 as shown in Figure 6.

Firstly, FlashFFTStencil demonstrates a substantial performance improvement over indirect stencil computations using the best implementation based on cuFFT and cuDNN, achieving performance gains ranging from a minimum of 1.9x to a maximum of 103.0x speedup. This enhancement is primarily attributed to the lack of specialized optimizations for stencil computations in these two methods. Secondly, FlashFFTStencil demonstrates an average speedup of approximately 5.8x over Brick and 2.9x over DRStencil. Lastly, compared to prior stencil methods on TCU, TCStencil, ConvStencil, and LoRAStencil, FlashFFTStencil achieves average speedups of

2.56x, 2.57x, and 2.44x, respectively. We will provide a more comprehensive analysis and comparison with these stencil methods using TCUs in the following subsection. It is noteworthy that, since TCStencil only supports FP6 precision for stencil computations, we employed the same evaluation methodology as used in ConvStencil [15, 74]. Additionally, given that LoRAStencil reduces the effective computational workload by 50% through the exploitation of kernel symmetry, we adjusted its execution time by multiplying a factor of 2.

### 5.4 Comparison with Stencil using TCU

In this subsection, we will perform a comparative analysis of FlashFFTStencil with existing stencil methods using TCU, including TCStencil, ConvStencil, and LoRAStencil. First, as illustrated by the data on the right vertical axis in Figure 10, we measured the sparsity of data within fragments during the execution on TCUs, defined as the ratio of the number of zeros value to the total number of data in the fragments. All of these methods demonstrate a sparsity of no less than 24.5%, which inevitably increases the memory workload to some extent. In contrast, FlashFFTStencil achieves fully dense TCU computations. Secondly, due to the bound-shifting effect of bridging FFT to stencil, a portion of the memory workload is converted into compute workload, significantly enhancing arithmetic intensity beyond that of ConvStencil and the turning point of A100 and H100. Furthermore, this heightened arithmetic intensity suggests that future GPUs with superior peak computational capabilities, such as the B100 [49], will yield even greater performance gains compared to other stencil methods.

## 6 Related Work

Research on accelerating stencil computations has been extensively explored on the different hardware architectures.

On CPUs [32, 71], vectorization techniques enhance parallelism in stencil computations by leveraging SIMD instructions [24, 25, 32]. Data reuse strategies effectively alleviate the memory bottleneck associated with stencil computations [58, 63, 75]. Additionally, cache optimizations and tiling are widely employed techniques for improving performance [8, 29, 69, 71]. There have been attempts to optimize stencil computations using FFT on CPUs [2–4, 35], but no such efforts have been studying on GPUs.

Extensive research has also been conducted on optimizing stencils on GPUs [40, 55, 59]. Tiling, a prevalent technique for optimizing stencil computations on GPUs, enhances parallelism and data locality [10, 18, 21, 26, 39, 43, 56, 67, 75]. Additionally, loop unrolling [22], prefetching [60], and streaming [59] techniques are widely used for optimizing stencil computations. Furthermore, TCStencil, ConvStencil, and LoRAStencil [15, 36, 74] represent recent efforts to leverage TCUs on GPUs. However, these approaches introduce significant sparsity into the computations, which can impact the overall performance of stencil calculations.

## 7 Conclusion

This paper presents FlashFFTStencil, a memory-efficient stencil computation system designed with the objective of bound shifting to bridge FFT for fully-dense stencil computations on TCUs. FlashFFTStencil is composed of three core techniques: Kernel Tailoring, Architecture Aligning, and Computation Streamlining. Results show that FlashFFTStencil achieves effective sparsity-free bound shifting, with a 2.57x average speedup over the state-of-the-art.

## References


[1] Hamdy Abdelkhalik, Yehia Arafa, Nandakishore Santhi, and Abdel-Hameed Badawy. 2022. Demystifying the Nvidia Ampere Architecture through Microbenchmarking and Instruction-level Analysis. arXiv:2208.11174 [cs.AR] https://arxiv.org/abs/2208.11174

[2] Zafar Ahmad, Rezaul Chowdhury, Rathish Das, Pramod Ganapathi, Aaron Gregory, and Yimin Zhu. 2021. Fast Stencil Computations using Fast Fourier Transforms. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures* (Virtual Event, USA) *(SPAA '21)*. Association for Computing Machinery, New York, NY, USA, 8–21. https://doi.org/10.1145/3409964.3461803

[3] 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 (dec 2023), 34 pages. https://doi.org/10.1145/3606338

[4] Zafar Ahmad, Mohammad Mahdi Javanmard, Gregory Croisdale, Aaron Gregory, Pramod Ganapathi, Louis-Noël Pouchet, and Rezaul Chowdhury. 2022. FOURST: A code generator for FFT-based fast stencil computations. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 99–108. https://doi.org/10.1109/ISPASS55109.2022.00010

[5] Yulong Ao, Chao Yang, Xinliang Wang, Wei Xue, Haohuan Fu, Fangfang Liu, Lin Gan, Ping Xu, and Wenjing Ma. 2017. 26 PFLOPS Stencil Computations for Atmospheric Modeling on Sunway TaihuLight. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 535–544. https://doi.org/10.1109/IPDPS.2017.9

[6] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. 2009. A View of the Parallel Computing Landscape. *Commun. ACM* 52, 10 (oct 2009), 56–67. https://doi.org/10.1145/1562764.1562783

[7] Krste Asanović, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report UCB/EECS-2006-183. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[8] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. 2012. Tiling Stencil Computations to Maximize Parallelism. In *Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)*. IEEE Computer Society, USA, 1–11. https://doi.org/10.1109/SC.2012.107

[9] Tal Ben-Nun, Linus Groner, Florian Deconinck, Tobias Wicky, Eddie Davis, Johann Dahm, Oliver D. Elbert, Rhea George, Jeremy McGibbon, Lukas Trümper, Elynn Wu, Oliver Fuhrer, Thomas Schulthess, and Torsten Hoefler. 2022. Productive Performance Engineering for Weather and Climate Modeling with Python. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. https://doi.org/10.1109/SC41404.2022.00078

[10] Uday Bondhugula, Vinayaka Bandishti, and Irshad Pananilath. 2017. Diamond Tiling: Tiling Techniques to Maximize Parallelism for Stencil Computations. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (May 2017), 1285–1298. https://doi.org/10.1109/TPDS.2016.2615094

[11] Victoria Caparrós Cabezas and Markus Püschel. 2014. Extending the roofline model: Bottleneck analysis with microarchitectural constraints. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 222–231. https://doi.org/10.1109/IISWC.2014.6983061

[12] S.C. Chan and K.L. Ho. 1991. On indexing the prime factor fast Fourier transform algorithm. *IEEE Transactions on Circuits and Systems* 38, 8 (1991), 951–953. https://doi.org/10.1109/31.85638

[13] Charlene Yang. 2018. Introduction to the Roofline Model. https://www.nersc.gov/assets/Uploads/Tutorial-ISC2018-Roofline-Model.pdf/, Last accessed on 2024-8-16.

[14] Peng Chen, Mohamed Wahib, Shinichiro Takizawa, Ryousei Takano, and Satoshi Matsuoka. 2019. A Versatile Software Systolic Execution Model for GPU Memory-Bound Kernels. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19)*. Association for Computing Machinery, New York, NY, USA, Article 53, 81 pages. https://doi.org/10.1145/3295500.3356162

[15] Yuetao Chen, Kun Li, Yuhao Wang, Donglin Bai, Lei Wang, Lingxiao Ma, Liang Yuan, Yunquan Zhang, Ting Cao, and Mao Yang. 2024. ConvStencil: Transform Stencil Computation to Matrix Multiplication on Tensor Cores. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Edinburgh, United Kingdom) *(PPoPP '24)*. Association for Computing Machinery, New York, NY, USA, 333–347. https://doi.org/10.1145/3627535.3638476

[16] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[17] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* 41, 2 (March 2021), 29–35. https://doi.org/10.1109/MM.2021.3061394

[18] Thomas L. Falch and Anne C. Elster. 2014. Register Caching for Stencil Computations on GPUs. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. 479–486. https://doi.org/10.1109/SYNASC.2014.70

[19] Daniel Y. Fu, Hermann Kumbong, Eric Nguyen, and Christopher Ré. 2024. FlashFFTConv: Efficient Convolutions for Long Sequences with Tensor Cores. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=gPKTTAfYBp

[20] I.J. Good. 1971. The Relationship Between Two Fast Fourier Transforms. *IEEE Trans. Comput.* C-20, 3 (1971), 310–317. https://doi.org/10.1109/T-C.1971.223236

[21] Tobias Grosser, Albert Cohen, Paul H. J. Kelly, J. Ramanujam, P. Sadayappan, and Sven Verdoolaege. 2013. Split Tiling for GPUs: Automatic Parallelization Using Trapezoidal Tiles. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units* (Houston, Texas, USA) *(GPGPU-6)*. Association for Computing Machinery, New York, NY, USA, 24–31. https://doi.org/10.1145/2458523.2458526

[22] Tobias Gysi, Christoph Müller, Oleksandr Zinenko, Stephan Herhut, Eddie Davis, Tobias Wicky, Oliver Fuhrer, Torsten Hoefler, and Tobias Grosser. 2021. Domain-Specific Multi-Level IR Rewriting for GPU: The Open Earth Compiler for GPU-Accelerated Climate Simulation. *ACM Trans. Archit. Code Optim.* 18, 4, Article 51 (sep 2021), 23 pages. https://doi.org/10.1145/3469030

[23] M. Heideman, D. Johnson, and C. Burrus. 1984. Gauss and the history of the fast fourier transform. *IEEE ASSP Magazine* 1, 4 (1984), 14–21. https://doi.org/10.1109/MASSP.1984.1162257

[24] Tom Henretty, Kevin Stock, Louis-Noël Pouchet, Franz Franchetti, J. Ramanujam, and P. Sadayappan. 2011. Data Layout Transformation for Stencil Computations on Short-Vector SIMD Architectures. In *Compiler Construction*, Jens Knoop (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 225–245.

[25] Tom Henretty, Richard Veras, Franz Franchetti, Louis-Noël Pouchet, J. Ramanujam, and P. Sadayappan. 2013. A Stencil Compiler for Short-Vector SIMD Architectures. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing* (Eugene, Oregon, USA) *(ICS '13)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/2464996.2467268

[26] Justin Holewinski, Louis-Noël Pouchet, and P. Sadayappan. 2012. High-Performance Code Generation for Stencil Computations on GPU Architectures. In *Proceedings of the 26th ACM International Conference on Supercomputing* (San Servolo Island, Venice, Italy) *(ICS '12)*. Association for Computing Machinery, New York, NY, USA, 311–320. https://doi.org/10.1145/2304576.2304619

[27] H.T. Huynh, Z.J. Wang, and P.E. Vincent. 2014. High-order methods for computational fluid dynamics: A brief review of compact differential formulations on unstructured grids. *Computers & Fluids* 98 (2014), 209–220. https://doi.org/10.1016/j.compfluid.2013.12.007 12th USNCCM mini-symposium of High-Order Methods for Computational Fluid Dynamics - A special issue dedicated to the 80th birthday of Professor Antony Jameson.

[28] Mathias Jacquelin, Mauricio Araya–Polo, and Jie Meng. 2022. Scalable Distributed High-Order Stencil Computations. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13. https://doi.org/10.1109/SC41404.2022.00035

[29] Guohua Jin, J. Mellor-Crummey, and R. Fowler. 2001. Increasing Temporal Locality with Skewing and Recursive Blocking. In *SC '01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*. 57–57. https://doi.org/10.1109/SC.2001.10041

[30] Jose Luis Jodra, Ibai Gurrutxaga, and Javier Muguerza. 2015. A Study of Memory Consumption and Execution Performance of the cuFFT Library. In *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. 323–327. https://doi.org/10.1109/3PGCIC.2015.66

[31] Hari Krishna. 2017. *Digital signal processing algorithms: number theory, convolution, fast Fourier transforms, and applications*. Routledge.

[32] Kun Li, Liang Yuan, Yunquan Zhang, and Yue Yue. 2021. Reducing Redundancy in Data Organization and Arithmetic Calculation for Stencil Computations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 84, 15 pages. https://doi.org/10.1145/3458817.3476154

[33] Kun Li, Liang Yuan, Yunquan Zhang, and Yue Yue. 2021. Reducing Redundancy in Data Organization and Arithmetic Calculation for Stencil Computations. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*. 01–15.

[34] Zhenyu Li, H. Sorensen, and C. Burrus. 1986. FFT and convolution algorithms on DSP microprocessors. In *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 11. 289–292. https://doi.org/10.1109/ICASSP.1986.1169089

[35] Song Liu, Xinhe Wan, Zengyuan Zhang, Bo Zhao, and Weiguo Wu. 2023. TurboStencil: You only compute once for stencil computation. *Future Gener. Comput. Syst.* 146, C (sep 2023), 260–272. https://doi.org/10.1016/j.future.2023.04.019

[36] Xiaoyan Liu, Yi Liu, Hailong Yang, Jianjin Liao, Mingzhen Li, Zhongzhi Luan, and Depei Qian. 2022. Toward accelerated stencil computation by adapting tensor core unit on GPU. In *Proceedings of the 36th ACM International Conference on Supercomputing* (Virtual Event) *(ICS '22)*. Association for Computing Machinery, New York, NY, USA, Article 28, 12 pages. https://doi.org/10.1145/3524059.3532392

[37] David J. Lusher, Satya P. Jammy, and Neil D. Sandham. 2021. OpenSBLI: Automated code-generation for heterogeneous computing architectures applied to compressible fluid dynamics on structured grids. *Computer Physics Communications* 267 (2021), 108063. https://doi.org/10.1016/j.cpc.2021.108063

[38] Tareq M. Malas, Julian Hornich, Georg Hager, Hatem Ltaief, Christoph Pflaum, and David E. Keyes. 2016. Optimization of an Electromagnetics Code with Multicore Wavefront Diamond Blocking and Multi-dimensional Intra-Tile Parallelization. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 142–151. https://doi.org/10.1109/IPDPS.2016.87

[39] Naoya Maruyama and Takayuki Aoki. 2014. Optimizing stencil computations for NVIDIA Kepler GPUs. In *Proceedings of the 1st international workshop on high-performance stencil computations, Vienna*. Citeseer, 89–95.

[40] Naoya Maruyama, Kento Sato, Tatsuo Nomura, and Satoshi Matsuoka. 2011. Physis: An implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1145/2063384.2063398

[41] Kazuaki Matsumura, Hamid Reza Zohouri, Mohamed Wahib, Toshio Endo, and Satoshi Matsuoka. 2020. AN5D: automated stencil framework for high-degree temporal blocking on GPUs. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*. 199–211.

[42] Robert Matusiak. 2002. Implementing Fast Fourier Transform Algorithms of Real-Valued Sequences With the TMS 320 DSP Platform. https://api.semanticscholar.org/CorpusID:11262963

[43] Jiayuan Meng and Kevin Skadron. 2009. Performance Modeling and Automatic Ghost Zone Optimization for Iterative Stencil Loops on GPUs. In *Proceedings of the 23rd International Conference on Supercomputing* (Yorktown Heights, NY, USA) *(ICS '09)*. Association for Computing Machinery, New York, NY, USA, 256–265. https://doi.org/10.1145/1542275.1542313

[44] Nvidia. 2023. NVIDIA A100 Tensor Core GPU Architecture. https://images.nvidia.cn/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, Last accessed on 2024-8-16.

[45] Nvidia. 2024. CUDA C++ Best Practices Guide. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html, Last accessed on 2024-8-16.

[46] Nvidia. 2024. CUDA C++ Programming Guide. https://https://docs.nvidia.com/cuda/cuda-c-programming-guide/, Last accessed on 2024-8-16.

[47] Nvidia. 2024. cuDNN. https://developer.nvidia.com/cudnn, Last accessed on 2024-8-16.

[48] Nvidia. 2024. cuFFT. https://docs.nvidia.com/cuda/cufft/index.html, Last accessed on 2024-8-16.

[49] Nvidia. 2024. NVIDIA Blackwell Platform Arrives to Power a New Era of Computing. https://nvidianews.nvidia.com/news/nvidia-blackwell-platform-arrives-to-power-a-new-era-of-computing, Last accessed on 2024-8-16.

[50] Nvidia. 2024. Parallel Thread Execution ISA Version 8.5. https://docs.nvidia.com/cuda/parallel-thread-execution/, Last accessed on 2024-8-16.

[51] Georg Ofenbeck, Ruedi Steinmann, Victoria Caparros, Daniele G. Spampinato, and Markus Püschel. 2014. Applying the roofline model. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 76–85. https://doi.org/10.1109/ISPASS.2014.6844463

[52] Oystein Ore. 1952. The General Chinese Remainder Theorem. *The American Mathematical Monthly* 59, 6 (1952), 365–370. https://doi.org/10.1080/00029890.1952.11988142 arXiv:https://doi.org/10.1080/00029890.1952.11988142

[53] Peter Van Sandt and Zhe Jia. 2021. Dissecting the Ampere GPU Architecture through Microbenchmarking. https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s33322/, Last accessed on 2024-8-16.

[54] Victor Podlozhnyuk. 2007. FFT-based 2D convolution. *NVIDIA white paper* 32, 1 (2007).

[55] 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. In *Proceedings of the*

*34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 519–530. https://doi.org/10.1145/2491956.2462176

[56] Prashant Rawat, Martin Kong, Tom Henretty, Justin Holewinski, Kevin Stock, Louis-Noël Pouchet, J. Ramanujam, Atanas Rountev, and P. Sadayappan. 2015. SDSLc: A Multi-Target Domain-Specific Compiler for Stencil Computations. In *Proceedings of the 5th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing* (Austin, Texas) *(WOLFHPC '15)*. Association for Computing Machinery, New York, NY, USA, Article 6, 10 pages. https://doi.org/10.1145/2830018.2830025

[57] Prashant Singh Rawat, Changwan Hong, Mahesh Ravishankar, Vinod Grover, Louis-Noël Pouchet, and P. Sadayappan. 2016. Effective Resource Management for Enhancing Performance of 2D and 3D Stencils on GPUs. In *Proceedings of the 9th Annual Workshop on General Purpose Processing Using Graphics Processing Unit* (Barcelona, Spain) *(GPGPU '16)*. Association for Computing Machinery, New York, NY, USA, 92–102. https://doi.org/10.1145/2884045.2884047

[58] Prashant Singh Rawat, Aravind Sukumaran-Rajam, Atanas Rountev, Fabrice Rastello, Louis-Noël Pouchet, and P. Sadayappan. 2018. Associative Instruction Reordering to Alleviate Register Pressure. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 590–602. https://doi.org/10.1109/SC.2018.00049

[59] 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. https://doi.org/10.1109/JPROC.2018.2862896

[60] Prashant Singh Rawat, Miheer Vaidya, Aravind Sukumaran-Rajam, Atanas Rountev, Louis-Noël Pouchet, and P. Sadayappan. 2019. On Optimizing Complex Stencils on GPUs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 641–652. https://doi.org/10.1109/IPDPS.2019.00073

[61] Gagandeep Singh, Alireza Khodamoradi, Kristof Denolf, Jack Lo, Juan Gomez-Luna, Joseph Melber, Andra Bisca, Henk Corporaal, and Onur Mutlu. 2023. SPARTA: Spatial Acceleration for Efficient and Scalable Horizontal Diffusion Weather Stencil Computation. In *Proceedings of the 37th ACM International Conference on Supercomputing* (Orlando, FL, USA) *(ICS '23)*. Association for Computing Machinery, New York, NY, USA, 463–476. https://doi.org/10.1145/3577193.3593719

[62] H. Sorensen, D. Jones, M. Heideman, and C. Burrus. 1987. Real-valued fast Fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35, 6 (1987), 849–863. https://doi.org/10.1109/TASSP.1987.1165220

[63] Kevin Stock, Martin Kong, Tobias Grosser, Louis-Noël Pouchet, Fabrice Rastello, J. Ramanujam, and P. Sadayappan. 2014. A Framework for Enhancing Data Reuse via Associative Reordering. *SIGPLAN Not.* 49, 6 (jun 2014), 65–76. https://doi.org/10.1145/2666356.2594342

[64] David Střelák and Jiří Filipovič. 2018. Performance analysis and autotuning setup of the cuFFT library. In *Proceedings of the 2nd Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems* (Limassol, Cyprus) *(ANDARE '18)*. Association for Computing Machinery, New York, NY, USA, Article 1, 6 pages. https://doi.org/10.1145/3295816.3295817

[65] Wei Sun, Ang Li, Tong Geng, Sander Stuijk, and Henk Corporaal. 2023. Dissecting Tensor Cores via Microbenchmarks: Latency, Throughput and Numeric Behaviors. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (Jan. 2023), 246–261. https://doi.org/10.1109/tpds.2022.3217824

[66] Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson. 2011. The pochoir stencil compiler. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures* (San Jose, California, USA) *(SPAA '11)*. Association for Computing Machinery, New York, NY, USA, 117–128. https://doi.org/10.1145/1989493.1989508

[67] Sven Verdoolaege, Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. 2013. Polyhedral Parallel Code Generation for CUDA. *ACM Trans. Archit. Code Optim.* 9, 4, Article 54 (jan 2013), 23 pages. https://doi.org/10.1145/2400682.2400713

[68] Wikipedia. 2024. Top500 Supercomputers. https://en.wikipedia.org/wiki/TOP500.

[69] David Wonnacott. 2002. Achieving Scalable Locality with Time Skewing. *Int. J. Parallel Program.* 30, 3 (jun 2002), 181–221. https://doi.org/10.1023/A:1015460304860

[70] Liang Yuan, Shan Huang, Yunquan Zhang, and Hang Cao. 2019. Tessellating Star Stencils. In *Proceedings of the 48th International Conference on Parallel Processing* (Kyoto, Japan) *(ICPP '19)*. Association for Computing Machinery, New York, NY, USA, Article 43, 10 pages. https://doi.org/10.1145/3337821.3337835

[71] Liang Yuan, Yunquan Zhang, Peng Guo, and Shan Huang. 2017. Tessellating Stencils. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17)*. Association for Computing Machinery, New York, NY, USA, Article 49, 13 pages. https://doi.org/10.1145/3126908.3126920

[72] Lingqi Zhang, Mohamed Wahib, Peng Chen, Jintao Meng, Xiao Wang, Toshio Endo, and Satoshi Matsuoka. 2023. PERKS: a Locality-Optimized Execution Model for Iterative Memory-bound GPU Applications. In *Proceedings of the 37th International Conference on Supercomputing*. 167–179.

[73] Lingqi Zhang, Mohamed Wahib, Peng Chen, Jintao Meng, Xiao Wang, Toshio Endo, and Satoshi Matsuoka. 2023. Revisiting Temporal Blocking Stencil Optimizations. In *Proceedings of the 37th International Conference on Supercomputing*. 251–263.

[74] Yiwei Zhang, Kun Li, Liang Yuan, Jiawen Cheng, Yunquan Zhang, Ting Cao, and Mao Yang. 2024. LoRAStencil: Low-Rank Adaptation of Stencil Computation on Tensor Cores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Atlanta, GA, USA) *(SC '24)*. IEEE Press, Article 53, 17 pages. https://doi.org/10.1109/SC41406.2024.00059

[75] Tuowen Zhao, Protonu Basu, Samuel Williams, Mary Hall, and Hans Johansen. 2019. Exploiting Reuse and Vectorization in Blocked Stencil Computations on CPUs and GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19)*. Association for Computing Machinery, New York, NY, USA, Article 52, 44 pages. https://doi.org/10.1145/3295500.3356210

[76] Tuowen Zhao, Mary Hall, Hans Johansen, and Samuel Williams. 2021. Improving Communication by Optimizing On-Node Data Movement with Data Layout. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 304–317. https://doi.org/10.1145/3437801.3441598

[77] Tuowen Zhao, Samuel Williams, Mary Hall, and Hans Johansen. 2018. Delivering Performance-Portable Stencil Computations on CPUs and GPUs Using Bricks. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. 59–70. https://doi.org/10.1109/P3HPC.2018.00009