Gamify Stencil Dwarf on Cloud for Democratizing Scientific Computing

Kun Li
Microsoft Research

Zhichun Li
HPC Research Center, CAS

Yuetao Chen
Microsoft Research

Zixuan Wang
HPC Research Center, CAS

Yiwei Zhang
HPC Research Center, CAS

Liang Yuan
HPC Research Center, CAS

Haipeng Jia
HPC Research Center, CAS

Yunquan Zhang
HPC Research Center, CAS

Ting Cao ✉
Microsoft Research

Mao Yang
Microsoft Research

Abstract

Stencil computation is one of the most important kernels in various scientific computing. Nowadays, most Stencil-driven scientific computing still relies heavily on supercomputers, suffering from expensive access, poor scalability, and duplicated optimizations.

This paper proposes Tetris, the first system for high-performance Stencil on heterogeneous CPU+GPU, towards democratizing Stencil-driven scientific computing on Cloud. In Tetris, polymorphic tiling tetrominoes are first proposed to bridge different hardware architectures and various application contexts with a perfect spatial and temporal tessellation automatically. Tetris is contributed by three main components: (1) Underlying hardware characteristics are first captured to achieve a sophisticated Pattern Mapping by register-level tetrominoes; (2) An efficient Locality Enhancer is first presented for data reuse on spatial and temporal dimensions simultaneously by cache/SMEM-level tetrominoes; (3) A novel Concurrent Scheduler is first designed to exploit the full potential of on-cloud memory and computing power by memory-level tetrominoes. Tetris is orthogonal to (and complements) the optimizations or deployments for a wide variety of emerging and legacy scientific computing applications. Results of thermal diffusion simulation demonstrate that the performance is improved by 29.6×, reducing time cost from day to hour, while preserving the original accuracy.

1 Introduction

Over the past decades, scientific computing is prosperously developed to solve Grand Challenges [46], such as solving genetic mysteries on Summit [44], exploring seismic simulation on Sunway Taihulight [18], and creating COVID-19 epidemic models on Fugaku [2], which utilize supercomputers to make significant progress.

Unfortunately, most scientific computing still relies heavily on traditional supercomputers [33, 19, 18], and it hinders the democratization of science by three main obstacles. First, the access to a supercomputer is luxurious for users, and it is even prohibitively expensive to employ a full machine for large-scale scientific computing. Second, duplication of efforts is made on them by engineers case by case due to diverse supercomputer architectures. Moreover, the scalability is poor as the assigned quota on a supercomputer is hard to adjust elastically.

Backed by advances in hardware and networking techniques, Cloud provides much higher performance scalability and cheaper access than traditional supercomputers. What is more, Cloud infrastructures, such as the ones from Microsoft Azure, Google Cloud, and Amazon Web Services [45, 52], widely adopt unified CPU and GPU heterogeneous architecture, driving the prosperity of AI. Therefore, Cloud contributes a promising unified, high-performance, low-cost, and elastic infrastructure for scientific computing.

However, none of the work supports scientific computing on Cloud for heterogeneous processors. The traditional techniques on supercomputers cannot be easily applied to the Cloud either, because of the significant differences in architecture. Although the AI surrogate models for scientific computing [28, 10, 48] could potentially leverage the cloud resources, the introduced precision loss is not afforded by diverse applications with the demanding taste of accuracy [5, 6, 33].

Though the applications are diverse in scientific computing, analogy to AI, there are several common and performance-critical operations in scientific computing, named *Dwarf*, defined by the Berkeley View [63]. As one of the seven computational Dwarfs, Stencil is ubiquitously involved in various scientific computing [14],
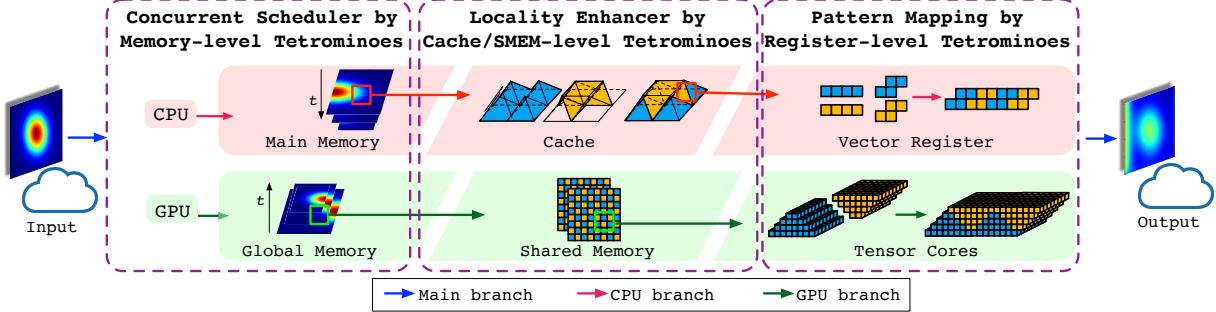

Figure 1: Tetris overview.

which lies at the heart of thermal diffusion ($\sim$100%), earth system model ($>$90%), and earthquake prediction model ($>$90%), etc [34, 59, 8, 40, 60, 15, 12].

To empower scientific computing on Cloud, we take Stencil-driven scientific computing as a case study and identify the following challenges.

Challenge #1: Complicated Spatial and Temporal Data Dependencies. Scientific computing is characterized by strong data dependencies in spatial and temporal dimensions since neighboring data are required to be collected to describe a phenomenon in a particular location and period of time, such as the computations of molecular motion, grid update, and interaction force among a variety of applications [33, 63, 18, 19]. In Stencil-driven scientific computing, the exhaustively studied tiling is one of the most powerful techniques to improve data reuse and performance [35, 63]. In general, they are designed for one type of memory on a specific hardware. However, there are different on-Cloud memory types for CPU and GPU, such as the registers and caches on the CPU and GPU, and the shared memory (SMEM) on GPU. One tiling shape cannot take full use of these resources, leading to severe performance degradation.

Challenge #2: Unfriendly On-Cloud Hardware Architecture. High-performance facilities such as Vector Registers on CPUs and Tensor Cores on GPU suffer from over specialization, as only specific operations are supported on them. However, the operators in scientific computing are highly diverse, and they cannot reap the benefits of these facilities trivially. On the one hand, though vectorization serves as an effective means of utilizing SIMD facilities on CPU, the data alignment conflicts are the main performance-limiting factors introduced by vectorization for scientific computing [24, 25], where the neighbors for a grid point appear in the same vector register but at different positions [35]. On the

other hand, Tensor Cores are specially designed for deep learning, and only simple matrix multiplication (MM) of specific size is supported [47, 13]. To the best of our knowledge, no previous work adopts Tensor Cores for scientific computing as it is prominently challenging to adapt abundant non-MM operations to the only one setting of FP64 MM operations on Tensor Cores [13].

Challenge #3: Essential CPU and GPU Collaboration. Most applications on the Cloud use CPU or GPU only due to the large performance difference. However, scientific computing calls for the collaborated CPU and GPU execution. Firstly, scientific computing requires huge memory costs. For example, the memory needed per simulated atom is approximately 0.72KB and a large-scale kinetic Monte Carlo (KMC) simulation system is contributed by nearly hundred-billion atoms [33]! It is very costly to use GPU memory only, which calls for the use of CPU memory. Secondly, different from AI computations, scientific computing requires vast FP64 operations. GPU has a similar performance as the CPU for this kind of computation, which calls for the collaborated execution of CPU and GPU. Third, the collaboration of CPU and GPU introduces increased communication which complicates the computing process, leading to bubbles in the pipeline.

Tetris. This paper proposes Tetris, the first system for high-performance Stencil on heterogeneous CPU+GPU, towards democratizing Stencil-driven scientific computing on Cloud.

The design of Tetris is based on two key observations.

First, changing data-update order would not break the application semantics. Specifically, Stencil aims to update data with restricted neighbors along the time dimension, while the updating order of data does not matter.

Second, creating a breakpoint would not destroy the computational logic. The dependencies of a point can

be decomposed and computed partially first, and the remaining dependencies are collected at a postponed time.

Guided by these observations, the principal insight is: the data are not necessarily aligned spatially or temporally in a regular order; similarly, the calculation of the current point in a time step is also allowed with a brief pause for tessellating other calculations.

Based on this insight, the key design of Tetris is *tiling tetrominoes tessellation*. The high-level idea is inspired by a classic Tetris game of sliding blocks (named tetrominoes). In a Tetris game, different tetrominoes are just kept piling onto old ones first to fill the vacated spaces, until they are aligned in a line. Analogously, with polymorphic tiling tetrominoes on different hardware hierarchies, Tetris carefully reorders the updating objects or marks a breakpoint on the partially-computed points, allowing a maximal updating is performed locally first while aligned spatially and temporally at a scheduled time point.

As a crucial thread in Tetris, tiling tetrominoes run through the whole memory hierarchy on heterogeneous processors of Cloud. Figure 1 illustrates that Tetris consists of three main components:

Pattern Mapping by Register-level Tetrominoes   Conventionally, only the workloads suitable for vectorization or MM operations are taken into account for offloading to specialized hardware units in existing work.

Tetris captures the underlying hardware characteristics and designs distinctive tetrominoes to perform a sophisticated pattern mapping from Stencil computation to vectorization and MM operations. Briefly, (1) Vector Skewed Swizzling is designed on Vector Registers, which utilizes skewed tetrominoes for building a conflict-free vectorized pipeline. It is the first design free of expensive cross-lane SIMD instructions (the CPU latency decreases from 3 to 1 per instruction [22]). (2) Tensor Trapezoid Folding is proposed on Tensor Cores, and it employs stair tetrominoes to adapt non-MM operations in Stencil as a series of reduction and summation operations. It can leverage the computing power of Tensor Cores for MM meanwhile preserving high accuracy (FP64).

Locality Enhancer by Cache/SMEM-level Tetrominoes
Instead of the conventional perspective that employs a single tiling shape to improve data locality, Tetris achieves an efficient locality enhancer with polymorphic tiling tetrominoes for data reuse on spatial and temporal dimensions simultaneously. In the design of locality enhancer, the coarse-grained tetrominoes from memory could be tessellated as an aggregate unit of tetrominoes on cache (or SMEM). Similarly, cache/SMEM-level tetrominoes can be decomposed as fine-grained

tetrominoes to registers. This flexible layout makes it possible for scientific computing adapted automatically to different architectures and application contexts simply by adjusting the number of data tetrominoes on cache (or SMEM).

Concurrent Scheduler by Memory-level Tetrominoes
Tetris proposes a novel concurrent scheduler designed specifically for scientific computing on the CPU and GPU. It identifies the implicit and explicit dependencies for Stencil workloads between CPU and GPU, and a two-way partitioning is performed on input to obtain memory-level tetrominoes. Concretely, (1) it explores a bidirectional design for squeezing memory consumption by maximizing memory savings on GPU while exploiting CPU memory for simulation. (2) To preserve compute efficiency, a balanced tetrominoes partitioning strategy is designed to achieve orders-of-magnitude equal computation on CPU compared to GPU, preventing the CPU compute from becoming a performance bottleneck. (3) Moreover, the communication of boundary tetrominoes between CPU and GPU is significantly reduced to enable better scalability.

Results   We have evaluated Tetris on varied classic benchmarks of Stencil Dwarf used for real-world applications. A thorough evaluation with Data Reorganization [64]), Auto Vectorization [35], Pluto [7], Folding [34], Brick [66] and AN5D [37] demonstrates that Tetris improves the performance by an overall of 4.4× and 2.8× on average compared to the state-of-the-arts (Data Reorganization and AN5D) and achieve a nearly linear scaling on Cloud. Furthermore, we employ Tetris for a simulation of thermal diffusion on a square copper plate where Stencil computations dominates overwhelmingly the whole simulation (∼100%) and it improves the performance of simulation by 29.6× **from day to hour** while preserving the **original accuracy**.

2   Background

2.1   Stencil Dwarf

Dwarfs are defined as a series of classic algorithmic methods by the Berkeley View [3], which earn the name by dragging down the performance in serving "Snow White" (scientific computing) due to their performance-critical operations.

Listed as one of the most important Dwarfs, Stencil is extensively employed in various scientific and engineering applications, and arises as a principal class of floating-point kernels in science [3, 4, 64]. Generally, Stencil contains a pre-defined pattern that updates each

point in $d-$dimensional spatial grid iteratively along the time dimension. The value of one point at time t is a weighted sum of itself and neighboring points at the previous time [55].

Listing 1 shows the Stencil of a regular Heat-2D (5 points) kernel, where $c1 \sim c5$ are the accumulation coefficients. Here we use one of the heat equations as a case study to explore Heat-2D Stencil introduced in Listing 1, and it models the physical transfer of heat in a region over time [40]. Equation 1 is the standard form in three dimensions, where $U(t,x,y)$ represent the heat at a point (x,y) at time t, and α is the thermal diffusivity.

$$\frac{\partial U(t,x,y)}{\partial t} = \alpha \left(\frac{\partial^2 U(t,x,y)}{\partial x^2} + \frac{\partial^2 U(t,x,y)}{\partial y^2} \right) \quad (1)$$

To obtain a numerical algorithm performed on computers, we discretize the dimensions x, y, and t into points spaced Δx, Δy, and Δt apart, which transforms a point (t,x,y) in continuous space as $(n\Delta t, i\Delta x, j\Delta y)$ in discretized space-time in Equation 2:

$$u_{i,j}^n \approx U(n\Delta t, i\Delta x, j\Delta y). \quad (2)$$

With the discretization and simplification of Equation 1 by using CFL number μ [40], the following update is yielded as:

$$u_{i,j}^{n+1} = (1-4\mu)u_{i,j}^n + \mu \left(u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n \right), \quad (3)$$

where Stencil is derived from heat equations in scientific computing.

2.2 Surrogate Models

More recently, there has been considerable progress and promise in the use of AI surrogate models to complex calculations like first-principles functions [28], which could utilize GPU to accelerate scientific computing and create potential possibilities for adapting to Cloud infrastructure. Albeit the use of AI surrogate models within scientific computing improves time-to-solution, most applications cannot utilize them due to the strict taste of accuracy and convergency, especially in the domains

---

Algorithm 1 Heat-2D Stencil.

Require: mesh A, coefficient $c1 \sim c5$.
 1: **for** $t = 0 \rightarrow T$ **do**
 2:     **for** $i = 0 \rightarrow N_x$, $j = 0 \rightarrow N_y$ **do**
 3:         $A[(t+1)\%2][i][j] = c1 \times A[t\%2][i][j] +$
             $c2 \times A[t\%2][i-1][j] + c3 \times A[t\%2][i][j+1] +$
             $c4 \times A[t\%2][i][j-1] + c5 \times A[t\%2][i+1][j]$
 4:     **end for**
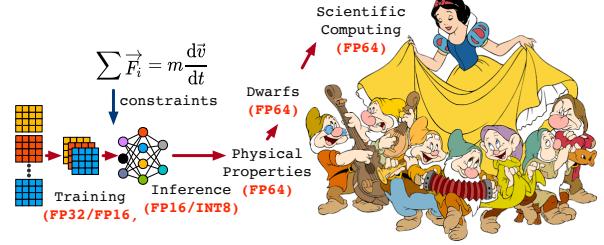 5: **end for**

Figure 2: AI surrogate models serving for scientific computing.

of weather forecast, galaxies collision, nuclear reactors, etc [5, 6, 34].

Figure 2 provides a schematic representation of utilizing AI surrogate models for scientific computing [28], which includes the three critical phases. First, AI surrogate models are designed and trained with simulation data. Once an appropriate trained model is built, it will replace partial calculations of physical properties. Second, the prepared physical properties are applied with Dwarfs for high-precision numerical simulation. At last, the numerical results produced by Dwarfs serve for "Snow White": the upper-level scientific computing, and a correction is promoted for potential biases or inaccuracies in trained models.

3 Pattern Mapping by Register-level Tetrominoes

With polymorphic register-level tetrominoes, Tetris designs a sophisticated *Pattern Mapping* to make full use of specialized hardware units on heterogeneous architecture.

3.1 Vector Skewed Swizzling

Existing vectorized optimizations for Stencil have mainly focused on either memory access or register permutation, aiming at decreasing transferred data volume and employing rich SIMD instructions respectively [34]. To address the data alignment conflicts [35] essentially, we perform a deep analysis of register architecture to find out the root cause of this challenge.

Vector Tetrominoes. Given a comprehensive view of registers in CPU, a 256-bit SIMD YMM register is actually promoted by two separate data lanes, where the lower 128-bits are aliased to a 128-bit XMM register [22]. Taking FP64 computations into consideration, we fill up this 256-bit SIMD register with 4 elements initially as a straight tetromino. Since a 256-bit SIMD reg-
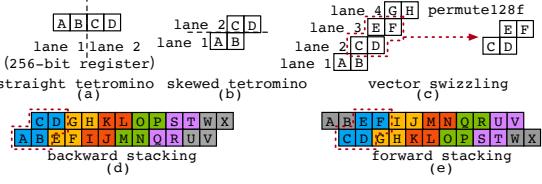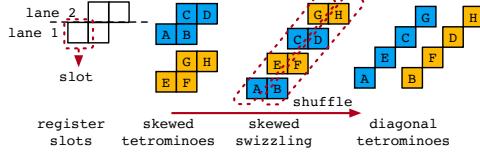
Figure 3: Skewed tetrominoes on CPU.

Figure 4: Skewed Swizzling in Vector Registers.

Figure 5: Quadruple Pipelining on Vector Registers.

Figure 6: Plane tetrominoes on GPU.

ister is composed of 2 128-bit register lanes, a straight tetromino is also viewed as an assembly by 2 bars accordingly in Figure 3(a). To adapt this lane-based characteristic of registers, variants are derived from straight tetrominoes to skewed tetrominoes by employing the 128-bit lanes as basic processing units.

As shown in Figure 3(b), a skewed tetromino constituted by 4 squares could hold a complete vector, which also indicates a 256-bit SIMD register is configured with 4 *double* type slots. Then we achieve *forward stacking* in Figure 3(d) by manipulating a crowd of skewed tetrominoes to assemble pipeline forward, and the whole data space for a time step could be tessellated flawlessly in this way. In view of the lane-based characteristic of registers, an equivalent conversion principle from forward stacking to *backward stacking* in Figure 3(e) is also designed swimmingly by a cheap lane-based `permute128f` instruction every two vectors illustrated in Figure 3(c).

Skewed Swizzling. The vectorization mechanism on CPU SIMD facilities is achieved by performing calculations on the slots of different registers vertically in one go, and each calculation flow on aligned slots is independent to each other. Thus, the key challenge to address the data alignment conflicts lies in an efficient adaption that remapping the adjacent elements to the same slot positions in different registers with minimal costs.

As illustrated preciously, lane-based operations are recommended for low-latency implementation when utilizing SIMD facilities on CPU. Figure 4 shows the process of skewed swizzling for two tetromino registers, where elements at the same slot position of each 128-bit lane are swizzled with elements in the other register correspondingly. Specifically, the two skewed tetrominoes A, B, C, D and E, F, G, H are swizzled for A, E,
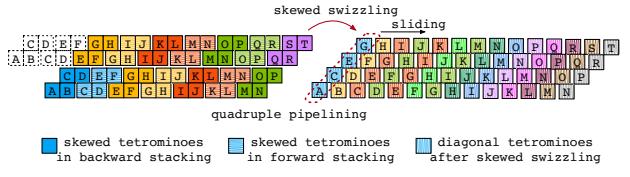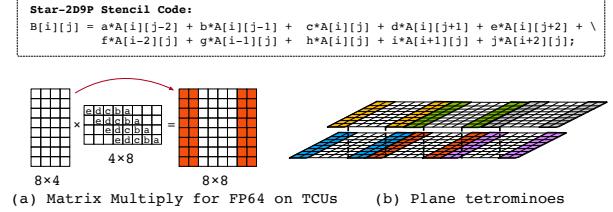
C, G and B, F, D, H by unpacking and interleaving elements from the low half and high half of each 128-bit lane respectively, which only consumes 1 latency time by lane-based `shuffle` instructions [20]. Based on the efficient skewed swizzling, two skewed tetrominoes are swizzled as diagonal tetrominoes, where the data alignment conflicts are all well addressed with minimal cost.

Quadruple Pipelining. Here we stack the forward and backward stacking in a well-designed principle to build a quadruple pipelining for a high-performance vectorized adaptation of Stencil Dwarf by Tetris.

Figure 5 (a) exhibits the principle of quadruple pipelining, where the forward stacking is tessellated with backward stacking tightly. More specifically, each skewed tetromino in forward stacking is inserted at intervals in backward stacking orderly to build a dual pipelining first. Then the head tetromino in each stacking is removed, and similar operations are performed again to obtain a new staggered dual pipelining. At last, we stack two dual pipelinings vertically and still keep a skewed style, and the two skewed tetrominoes are further converted to diagonal tetrominoes with a simple skewed swizzling in Figure 5 (b). With polymorphic tetrominoes, Tetris adapts Stencil Dwarf efficiently to CPU SIMD facilities by relieving data alignment conflicts in vectorization.

3.2 Tensor Trapezoid Folding

Tensor Tetromino. From NVIDIA Ampere GPU architecture like A100, the newly-introduced specialized
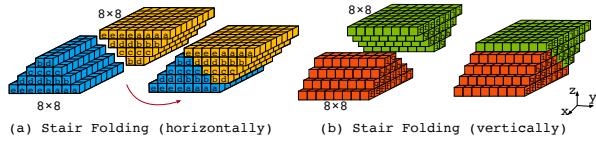
Figure 7: Tensor Trapezoid Folding on Tensor Cores.

Figure 8: Octuple Pipelining on Tensor Cores (FP64).

hardware units known as NVIDIA's Tensor Cores could offer a full range of precision to boost the performance of MM even with the highest accuracy needed (FP64). Although Tensor Cores are prevalent and promising to provide an increase in performance, they suffer from over specialization as only MM on specific size of small matrices is supported. In particular, the size of supported matrix for FP64 precision is only constrained to one setting shown in Equation 4:

$$D_{m \times k} = A_{m \times n} \times B_{n \times k} + C_{m \times k}, \qquad (4)$$

where m, n, k is 8, 4, 8 respectively.

Here a series of tensor tetrominoes are designed to achieve an efficient Stencil Dwarf adaptation. Concretely, at each step of Stencil, it applies the parameter kernels onto a portion of the elements in the specific shape, compute their element-wise products, and further sum the products together to update the value at this position. With this in mind, we could actually formulate the Stencil computation by using MM operations equivalently. Then Stencil is adapted as weighted reductions for a 8×8 plane tetromino.At last, the adjacent plane tetrominoes are accumulated with overlaps for the final update at this position.

Trapezoid Folding. A plane tetromino is an abstraction of the matrix C described from a two-dimensional perspective. To make different weights counts, we upgrade the plane tetrominoes as stair tetrominoes from a three-dimensional perspective. Then a Trapezoid Folding strategy is presented to build a tessellation of data space by utilizing Stair tetrominoes as building blocks.

Figure 7(a) exhibits a sketch of Trapezoid Folding. Each position in a $xy-$plane tetromino rises prominently towards the z dimension, building a symmetrical stair tetromino. The height is determined by the weights contributions from the parameter matrix B, and the positions in the same column are also of equal height due to the identical B used in MM operations. To distinguish it more intuitively, the stair tetrominoes are inverted and colored alternatively, and then they are stacked to tessellate the whole data space. We call the calculation along the third dimension with two stair tetrominoes a *folding*. For example, a folding is performed first on the right part of stair tetrominoes A (blue) with the left part of adjacent stair tetrominoes B (yellow). Then the overlapped positions are all updated horizontally, where the heights are
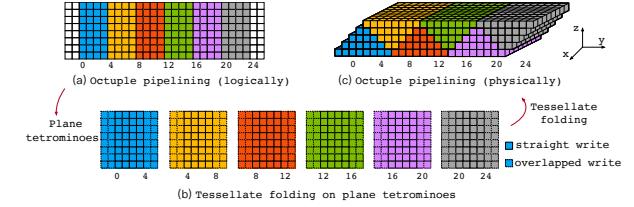
accumulated with 5 weights from all dependent neighbors. Similar operations are performed for obtaining a vertical folding in Figure 7(b).

Octuple Pipelining. Extended from vector to tensor, the tetrominoes defined on GPU are manipulated with a 8×8 matrix in Tensor Core fragments. Since the elements of the same column in matrix C are obtained by identical parameters in matrix B, the computations are performed along the x-direction horizontally, and each row represents a thread of calculation flow.

Based on the efficient Trapezoid Folding and Checkerboard Blocking, we achieve Octuple Pipelining for Stencil in Figure 8. First, the scientific data inputs on Checkerboard in shared memory are loaded into fragments as a 8×4 matrix with different colors in Figure 8 (a). Then stair tetrominoes collecting dependencies of the adjacents are built and stacked by Trapezoid Folding to construct Octuple Pipelining. As shown in Figure 8 (c), an efficient Octuple Pipelining is built to tessellate the whole data space, which achieves a high-precision adaptation from Stencil Dwarf to MM operations on Tensor Cores.

4 Locality Enhancer by Cache/SMEM-level Tetrominoes

In this subsection, we will introduce the *Locality Enhancer* by tetrominoes of the middle layer (cache on CPU and shared memory on GPU) in memory hierarchy, which enhances the data reuse spatially and temporally meanwhile matching other tetrominoes in top and bottom memory hierarchies.

4.1 Tessellate Tiling

Since modern high-performance CPU is generally configured with multiple cores, the data space can be tessellated spatially to enable fully parallelism such that all tetrominoes between synchronizations can execute concurrently without redundant computation. To develop more data reuse with fine optimizations, temporal tiling is then also presented by stretching the tetrominoes along the time dimension additionally.
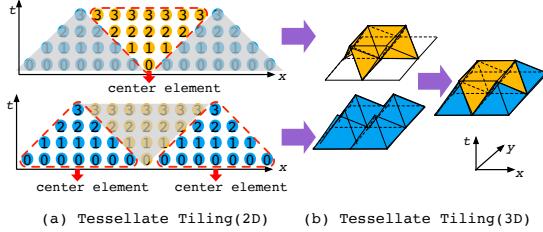
(a) Tessellate Tiling(2D)   (b) Tessellate Tiling(3D)

Figure 9: Tessellate Tiling ($T_b = 3$). Triangle tetrominoes are tessellated in 2D and upgrade as tetrahedron tetrominoes in 3D.

Figure 10: Checkerboard Blocking on shared memory.

Typically, the working sets are many orders of magnitude larger than a processor's cache memory. Let $T_d\{m,n\}$ be the data transfer time through the memory hierarchy m to n, and we use this shorthand notation for expressing data transfer time in each iteration (e.g., $T_d\{L2, Reg\} = T_{L2L1} + T_{L1Reg}$ when the data is in the L2 cache). Since all grid cells are updated once before the next update sweep starts, a bad data locality is common to Stencil Dwarf with $T_d\{Mem, Reg\} = T_{MemL3} + T_{L3L2} + T_{L2L1} + T_{L1Reg}$, where $T_{MemL3} \gg T_{L3L2} > T_{L2L1} > T_{L1Reg}$.

Spatial Tiling. Tessellate Tiling is used first by employing shaped tetrominoes to reduce $T_d\{Mem, Cache\}$, which dominates a major cost of data transfer time [63, 34]. It can be viewed as a tessellation in iteration space by utilizing shaped tetrominoes. Figure 9 illustrates the tiling strategy for a two-dimensional Stencil computation with $T_b = 3$ (temporal tiling size), where the timestamp of each element along the time dimension is annotated on it. Triangle tetrominoes are updated first, where the center element is updated three steps and its neighbors are updated fewer steps proportional to the distance with the center element. Then two half parts from adjacent triangle tetrominoes constitute new inverted triangle tetrominoes, and identical operations are performed to make all elements updated with the same steps. From the perspective of front view, The iteration space is tessellated by triangle tetrominoes and inverted triangle tetrominoes in alternative stages.

Temporal Tiling. As shown in Figure 9, concurrent execution is processed by two stages over the same steps temporally with spatial tessellation completed, and the updated step of a specific element in a tetrominoes is proportional to its distance with the center element. Taking updated steps into account, the 2D iteration space in Figure 9 rises straight from the ground. At this point, the 2D triangle tetrominoes are also expanded as 3D tetrahedron tetrominoes, and two types of them tessellate the whole data space along the spatial and temporal dimensions.
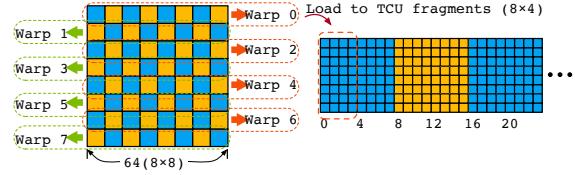
With the tessellate tiling strategy, concurrent execution for different tiles is enabled over a given time range without redundant computation [63], and a high in-memory flops/byte ratio is achieved with more data reused.

4.2 Checkerboard Blocking

An efficiently optimized GPU code must allocate appropriate GPU memory resources, especially on shared memory, to improve data locality. Most code generators automatically determine this resource assignment (or mapping), but without considering the balance on resource limits of the underlying GPU device and the requirements of applications. Taking the warp size (32 threads) and SMEM capacity into account, Checkerboard Blocking is carefully designed to provide an enhanced locality manually.

Spatial Blocking. As a bridge for connecting global memory and registers, the shared memory is utilized as the stored buffer to guarantee the coalesced global memory access, where the computed results from registers will also be stored back to it first. As shown in Figure 10, Checkerboard Blocking strategy is proposed for utilizing the shared memory efficiently. To achieve a high parallelism, the shared memory in a GPU block is abstracted as an 8×8 checkerboard that is criss-crossed by two types of square tetrominoes. Each row on the checkerboard is computed by a warp, and the threads in a warp are responsible for each concrete square tetromino. With Checkerboard Blocking, the data space on shared memory is occupied spatially without leaving any memory space unexploited or computing threads idle, and the coordinated data exchange could be performed with coalesced writes to global memory.

Conflict-free Blocking. It is required to address two major different conflicts on shared memory for unlocking a higher performance. One is the bank conflict triggered by hardware mechanism, and the other is block conflict introduced by inter-block compute dependencies. Instead of padding the shared memory buffer with additional columns, Checkerboard Blocking reduces the read and write bank conflicts entirely by a precise control of thread access. Specifically, each square tetromino on
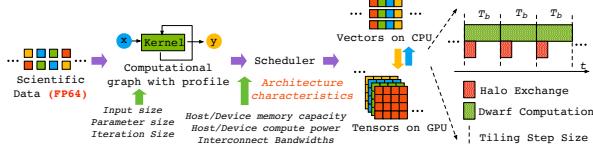
Figure 11: Concurrent heterogeneous scheduler.

checkerboard is further decomposed into an 8×8 grid, and a grid point represents a real data. Then the half part of tetromino with 8×4 grid points is loaded into a fragment on Tensor Cores, where the data and threads are mapped with a biunique correspondence without any conflict. As for block conflicts, Checkerboard Blocking stagger the square tetrominoes with two colors in Figure 10, and the updates are performed alternatively by warp scheduling and thread controlling precisely to avoid inter-block compute dependencies.

5 Concurrent Scheduler by Memory-level Tetrominoes

Tetris identifies the implicit and explicit dependencies between CPU and GPU, and a two-way partitioning is then performed on input as memory-level tetrominoes. Towards an efficient *Concurrent Scheduler* on heterogeneous processors, the following techniques are proposed on memory-level tetrominoes.

5.1 Bidirectional Memory Squeezing

Memory consumption is still a tough challenge for scientific computing, especially in large-scale high-precision simulation [50, 33]. Unlike previous work only utilizing CPU memory for high-precision simulation, Bidirectional Memory Squeezing in Tetris is designed to enable more efficient simulation by sharing memory resources on CPU and GPU.

The scientific computing workloads can be represented as a directed graph of data and computation. Figure 11 shows a Stencil Dwarf graph, where the output in the current step will be used as input for the next step. Since the spatial dependencies are much weaker than temporal dependencies, the offload strategy between GPU and CPU can then be abstracted by employing a two-way partitioning of input, such that tetrominoes in a partition would be executed and stored on that compute worker to explore more memory quota. It is also worth noting that the strategy is achieved with little offload cost for more computing and storage benefits.

Bidirectional Memory Squeezing is also a bidirectional design for squeezing memory consumption. Once the GPU memory is fully occupied, the remaining part left on CPU is still well-addressed since both CPU and GPU provide memory and compute for better efficiency. When multiple GPUs are available, Bidirectional Memory Squeezing could be extended similarly to offer excellent scalability in even larger scientific computing applications.

5.2 Auto-tuning Computation Scheduling

Tetris treats scientific computing applications as decomposable workloads since tetrominoes are coordinated explicitly by it, with each type of worker (CPU or GPU) executing a subset of the tetrominoes. Like with any distributed system, the steady state throughput of the whole heterogeneous architectures is determined crucially by the slowest worker. Having CPU and GPU process subtask at vastly different throughput can lead to bubbles in the computing pipeline, starving the faster worker of subtasks to work on and resulting in resource underutilization.

Tetris sticks to the fact that Stencil Dwarf exhibits little variance in computation time across the same size of inputs. As shown in Figure 11, in the startup phase, the input stage admits enough tetrominoes to keep each worker full in steady state. Then the computation time taken by the first iteration, the input size of scientific data, the size of Dwarf parameters and the number of iterations are recorded as part of a profile initialization. This profile is employed as the input to the scheduler for performing a balanced partition on a heterogeneous platform. Apart from the constraints of software, the scheduler is also architecture-aware in a design of taking into account other hardware characteristics, such as host/device memory capacity, host/device compute power, and the interconnect bandwidths, and it computes: (1) a partitioning on the input into workers, (2) the estimated communication or replication volume between workers, and (3) optimal number of in-flight tetrominoes to keep the computing pipeline busy.

Intuitively, this process finds the optimal partitioning on CPU and GPU and can also be extended easily across clusters. Based on the partitioning strategy, Tetris's scheduler dispatches a balanced workload to each worker on the main memory/global memory level.

5.3 Minimized Communication Cost

Less Communication Volume. Most of the existing work employing GPU accelerators requires a busy data transfers between CPU and GPU, as the CPU only plays a role of stream controlling and data is updated between CPU and GPU round by round. Tetris can communicate far less than this. Instead of having to perform a memory turnover periodically, each worker first deals with the halo region introduced by balanced computa-

tion scheduling, and has to communicate only this small subset of the working set.

More Communication Overlap. Despite using a highly reduced halo exchange, the communication interlude can still become a bottleneck that disturbs the regular computing pipeline. For such limited cases, we separate the streams of computation and communication carefully for a parallel execution, and address halo updates first in each working set. As shown in Figure 11, with the halo region updated, the exchange is performed immediately, which overlaps computation and communication to hide the halo exchange overhead.

Centralized Communication Launch. For the communication part, we collect the halo exchanges for T_b steps and then only conduct one centralized communication instead of conducting T_b small communications.The reason is analyzed below. The time for communication could be formulated by $k \times (\alpha + n_b \times \beta)$, where k is the number of messages; α is launch latency; $1/\beta$ is the bandwidth; n_b is the number of bytes per message. In practice, since we have $\alpha >> \beta$, an intact big message is much cheaper than k split small messages due to $k \times (\alpha + n_b\beta) >> (\alpha + k \times n_b \times \beta)$. Thus, centralized communication launch are proposed to further decrease communication cost for T_b steps.

6 Evaluation

6.1 Experimental Setup

Machine Experimental results presented in this paper are obtained by using a high-performance node on Microsoft Azure, which is composed of an AMD EPYC 7V13 processor with 24 physical cores of 2.45 GHz clock speed. It features the AVX2 SIMD instruction set, and each core contains a 768 KB private L1 cache, a 12 MB private L2 cache, and a unified 96 MB L3 cache. DDR4 DRAM and 8 memory channels are supported for a total 216 GB memory, and it yields a peak memory

Table 1: Configuration for Stencil benchmarks

Type	Pts	Problem Size	Blocking Size
Heat-1D	3	10,000,000×100,000	2,000×1,000
Star-1D5P	5	10,000,000×100,000	2,000×500
Heat-2D	5	10,000×10,000×10,000	200×200×50
Star-2D9P	9	10,000×10,000×10,000	200×200×50
Box-2D9P	9	10,000×10,000×10,000	2,000×500
Box-2D25P	25	10,000 ×10,000×10,000	120×128×60
Heat-3D	7	1,024×1,024×1,024×1,000	20×20×10
Box-3D27P	27	1,024×1,024×1,024×1,000	20×20×10

bandwidth of 409.6 GBps. An Nvidia A100 GPU is also configured with 80 GB memory, and the memory bandwidth could reach 1,935 GB/s. The most characteristic Tensor Cores with 108 SMs deliver a peak FP64 throughput of 19.5 TFLOPS, which is 2.5x that of Nvidia V100.

State-of-the-arts Although recent studies on Stencil Dwarf only exhibit their own absolute performance without comparison in experiments [37, 66, 1], we reproduce artifacts exhaustively and compare the performance of different state-of-the-arts for a comprehensive analysis. Two classic vectorization methods (Auto Vectorization [35] and Data Reorganization [64]) are employed first as a standard baseline on CPUs. Then the newly-related state-of-the-arts (Tessellation [63] and Folding [34]) are adopted for further comparison. At last, the highly-optimized work with GPU support, Brick [66] and AN5D [37], are also evaluated thoroughly. We utilize the OpenMP pragma *parallel for* on all methods for scalability experiments.

Benchmarks We use a variety of Stencil kernels with different shapes as benchmarks. Details are listed in Table 1, which consists of five star kernels (Heat-1D, 1D5P, Heat-2D, Star-2D9P, and Heat-3D) and three box kernels (Box-2D9P, Box-2D25P, and Box-3D27P) drawn from [64, 34, 7].

Programming Provided with a mathematical look into various benchmarks, we abstract the application skeletons as a semi-automatic code generator with Python wrapper. For programming on CPU, we use C/C++ for Tessellate Tiling and AVX2 SIMD instruction set for Vector Skewed Swizzling [22]. Then Checkerboard Blocking is programmed with CUDA C/C++ and Tensor Trapezoid Folding is coded in a manner of MM by Warp Matrix Multiply and Accumulate (WMMA) API [43].

Metrics Most work on Stencil Dwarf (e.g., Tessellation [63, 64], Pluto [7], Folding [35, 34], etc.) exhibit results in terms of arithmetic performance (Stencils/s). In this work, we also adopt the metric of stencils per second (Stencils/s) defined in Equation 5 for measuring the performance. Here, N_x, N_y, N_z are the problem size for each spatial dimension; T is the iteration item in temporal dimension; *time* is the total execution time.

$$\text{stencils per second} = \frac{N_x \cdot N_y \cdot N_z}{time} \times T \qquad (5)$$

6.2 Performance Breakdown

In this subsection, we investigate how the performance is improved progressively with different optimizations by Tetris.
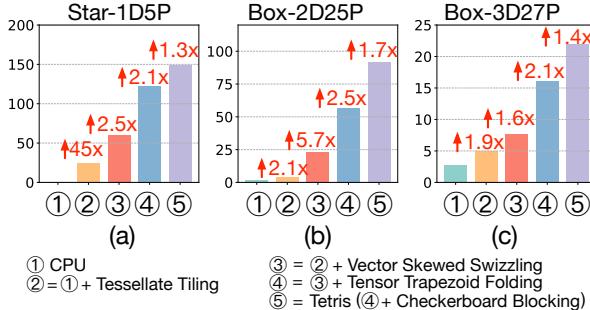
Figure 12: Performance breakdown of Tetris.

Table 2: Technical overview of state-of-the-arts and Tetris

Methods	Tiling	Pipeling	Architectures
Data Reorg. [64]	Split	Data Reorg.	C(VR)[1]
Auto Vec. [35]	Split	Auto Vec.	C(VR)
Pluto [7]	Diamond [9]	AutoVec.	C(VR)
Folding [34]	Polyhedral	Folding	C(VR)
Tetris (CPU)	Tessellate	Skewed[2]	C(VR)
Brick [66]	Brick	Scatter	G(CCU)
AN5D [37]	Temporal	-	G(CCU)
Tetris (GPU)	Checkerboard	Trapezoid	G(TCU)
Tetris	Polymorphic	Template	C(VR) + G(TCU)

[1] For better clarity, CPU, GPU, Vector Register, CUDA Core Units and Tensor Core Units are abbreviated with C, G, VR, CCU and TCU respectively.
[2] Notice we use a keyword as an abbreviation for some algorithms.

Figure 12 shows the performance breakdown of Tetris on three representative benchmarks (Star-1D5P, Box-2D25P, and Box-3D27P) as they contain more complex data access patterns and pose greater performance demands of Tetris.

As can be seen from Figure 12, optimizations on CPU in Tetris are first investigated on three benchmarks. Tessellate Tiling enhances the data reuse in CPU cache effectively and even provide a 45x speedup considerably in Star-1D5P benchmark. Next, Vector Skewed Swizzling utilizing the high-performance SIMD facilities in CPU is performed and apparently outperforms the previous version in all experiments. Since Vector Skewed Swizzling reduces redundant loading costs and minimizes the intra- & inter- register operations, it could relieve data alignment conflicts in vectorization efficiently and provide significant speedups from 1.6x to 5.7x. Here we have shown that a nerfed Tetris with multicore CPU could reach 112.5x, 12.0x, 3.1x speedups respectively on these representative benchmarks.

Then GPU is introduced in Tetris for further performance improvements. With the deep exploitation of Tensor Cores, all benchmarks achieve a minimum of doubled performance compared to the nerfed Tetris on CPU. Like CPU cache, shared memory provides more efficient data retrieves but limited capacity between local register and global memory. After utilizing shared memory by Checkerboard Blocking, 1.3x-1.7x speedups are obtained compared to the previous version. Up to this point, Tetris has accumulatively achieved speedups of 307.1x, 50.9x, 9.0x respectively, showing significant performance benefits brought by Tetris on multidimensional benchmarks.

6.3 State-of-the-art Comparison

In this subsection, we present the experiments that exhibit the benefits of Tetris with various state-of-the-arts on different benchmarks, and the technical overview of them are listed in Table 2.

As shown in Figure 13, taking all benchmarks with

AVX2 instructions on CPU into account, remarkable performance improvements are observed from Tetris (CPU). Compared with the newly-released state-of-the-art Folding, Tetris (CPU) improves the overall performance sustainably by an average of 21% and a maximum of 25% in Box-2D9P and Box-3D27P benchmarks, which demonstrates that our optimization on CPU architecture could provide a significant benefit especially in complex pattern compared to the referenced work.

Here, to evaluate whether a nerfed Tetris only run on GPU can achieve a competitive performance, we additionally employ another two state-of-the-arts for a further comparison, where AN5D is a highly-optimized work using CUDA Cores. Apparently, Figure 13 shows Tetris (GPU) achieves the best for all benchmarks even compared to the previous CPU baselines. Compared to AN5D, Tetris (GPU) achieves an overall 1.9x speedup on average and even reach a maximum 2.2x speedup in Figure 13(f)! The reason is that, the Dwarf computation on Tensor Core is adapted efficiently by Tensor Trapezoid Folding, and Checkerboard Blocking make full use of shared memory for data reuse especially more friendly to complex benchmarks.

When Tetris is performed on CPU and GPU, the performance potential is fully expressed and it runs rings around all state-of-the-arts. We still observe that the Tetris performance is arithmetically close to the sum of two nerfed versions (Tetris (CPU) and Tetris (GPU)) for all benchmarks. This is because the cost of communication between CPU and GPU is greatly reduced and overlapped with computation, and the auto-tuning scheduling mechanism provides a balanced task partition, which make the utmost of computing power on Cloud. Compared to Data Reorganization, Tetris improves the overall performance by an average of 4.4x and a maximum of 8.1x in Box-2D25P.
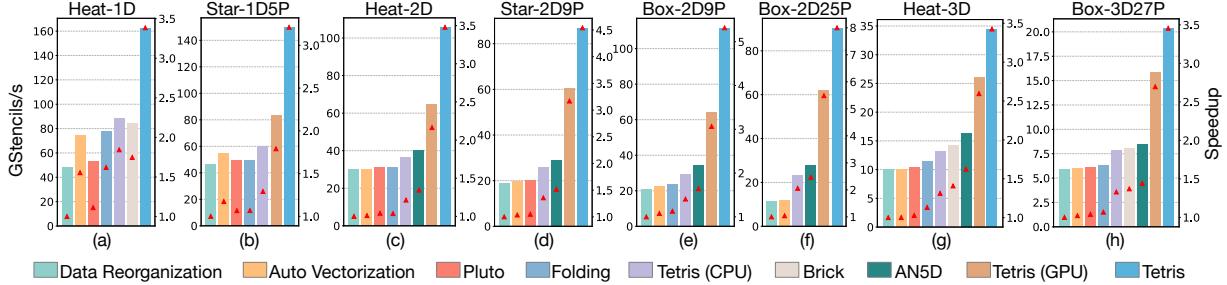
Figure 13: Performance and speedup comparison of state-of-the-arts and Tetris. The speedups of each group are compared to the lowest base which is annotated with the triangles by default value of 1.0.
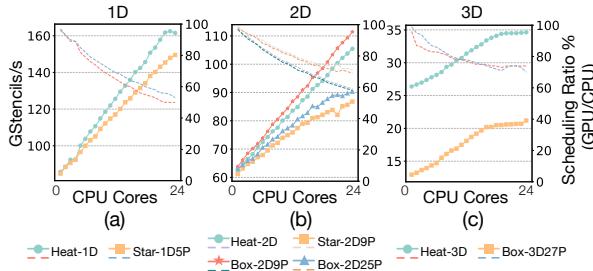

Figure 14: Scalability for Tetris with various benchmarks. Solid lines with markers represent the performance, and scheduling ratios (GPU to CPU) of task partition are illustrated with dotted lines.

6.4 Scalability Evaluation

We also evaluate the scalability of Tetris on Cloud for a comprehensive analysis. The detailed parameters are given in Table 1, where all problem sizes are large enough to approach the scale in a real simulation.

It can be observed from Figure 14 that Tetris could achieve a good scaling with increased CPU cores. In 1D and 2D benchmarks, nearly linear scalings on Cloud are obtained by Tetris and the CPU with 24 cores provides a competitive performance to GPU in the same order of magnitude. Surprisingly, a 49.9% scheduling ration is determined by Tetris, which demonstrates a scalable scheduler design and breaks the stereotype of CPU inferior to GPU. With the increase of the benchmark dimension, the scalability drops slightly when more CPU cores are employed due to the inherent complexity for multidimensional computations. Moreover, it has to perform a great deal of coordination among CPU cores, incurring communication overheads inevitably.

6.5 Case Study: Thermal Diffusion

To further prove the validity of Tetris in real application, simulations of thermal diffusion on a square copper plate (CFL number: $\mu = 0.23$) are performed at

Figure 15: Code snippets for thermal diffusion simulation with Tetris.

Table 3: Performance improvements by Tetris

Methods	Time(s)	Performance (GStencil/s)	Speedup
Naive	124,448.5	2.8	-
Tetris (CPU)	27,709.1	14.8	4.5x
Tetris (GPU)	5,597.4	63.3	22.6x
Tetris	4,270.9	82.9	29.6x

$100°C$ with dimensions of 15 mm on a side (Grid size: $9,600 \times 9600$). The simulation is to calculate the temperature distribution D depending on time for a long-time evolution (3.8×10^6 time steps), i.e. $D = U(t,x,y)$ simulated by Laplacian numerical calculation using 5-point Stencil finite difference methods in Section 2.1.

Provided with a simple interface illustrated in Figure 15, scientists could democratize large-scale thermal simulation on Cloud with minimal efforts by Tetris. Table 3 shows the detailed performance improvements on thermal diffusion by Tetris. The speedup ranges from 4.5x to 29.6x with different versions of Tetris, showing that it could provide a significant benefit over real scientific computing applications.

Table 4: Analytical accuracy comparison[1]

Deviation	Absolute Error (°C)			Relative Error (%)		
	>0.1	>0.5	>1.0	>1.0	>3.0	>5.0
Tetris by FP64 (%)	0.0	0.0	0.0	0.0	0.0	0.0
Naive by FP32 (%)	73.1	17.2	11.3	62.1	14.3	0.2

[1] The accuracy errors are all compared with the Naive method(FP64).
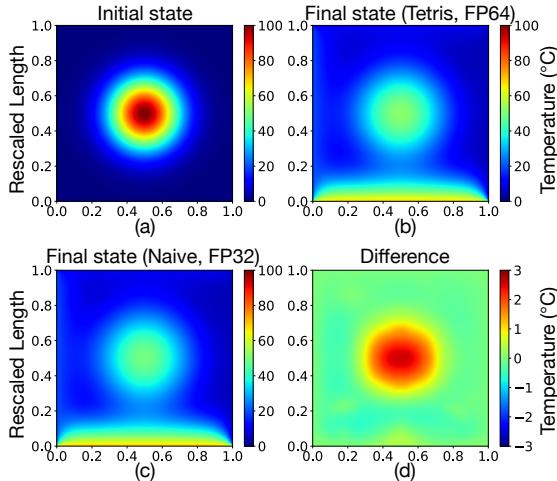
11

Figure 16: Visualization of temperature distribution before and after the thermal diffusion on the square Cu plate. The difference is measured by absolute errors (the reds: positive errors (hotter), the greens: zero errors, the blues: negative errors (colder)).

As shown in Figure 16(a), at the beginning of the simulation, it is clear that the initial temperature is a Gaussian, and it will be cooler at the edges with the hottest temperature at the plate center. After a long-time thermal evolution by Tetris, the temperature distribution in Figure 16(b) is radically different from the one obtained at the beginning, and the temperature of plate center decreases to 52.0°C, which is coherent with the thermodynamic theory [8].

It is worth noting that we also perform a comparison experiment with FP32 on the same initial state of the system. The results by FP32 and the differences with Tetris (FP64) are visualized in Figure 16(c) and Figure 16(d) respectively. Table 4 also lists an analytical accuracy comparison of FP64 and FP32 intuitively. Significantly, an overall 73.1% deviation is measured for temperatures fluctuated by 0.1°C, which demonstrates an unignorable variation caused by insufficient accuracies (FP32).

7 Related Work

Known as one of the seven Dwarfs, Stencil is extensively involved in scientific computing [34, 3]. The representative work can largely be classified in two directions in terms of architectures.

It is acknowledged that scientific computing is sensitive to the data precision, and thus a large quantity of work addresses this issue by utilizing FP64 operations on CPU architectures. Traditional approaches have mainly focused on either vectorization or tiling schemes, aiming at improving the in-core data parallelism and the

data locality in cache respectively [63]. For SIMD implementation on CPU, data alignment conflicts incurred by vectorization is a crucial performance-limiting factor [35, 24, 66]. Bandishti et al. enhance the Pluto compiler to incorporate a strategy for diamond tiling, and is particularly effective in parallelizing stencil computations [7]. Folding reduces the data reorganization overhead for vectorization and shows a better performance than Pluto [7], Tessellation [63], YASK [61, 62] and DLT [25], which is considered as one of state-of-the-arts approach for vectorization. However, the frequent in-CPU transpose easily incurs register spilling [34]. Tiling [27, 38, 32, 56, 57] is one of the most powerful transformation techniques to explore the data locality of multiple loop nests. Notable work for Stencil includes hyper-rectangle tiling [16, 49, 51, 41], time skewed tiling [53, 58, 29], diamond tiling [9, 7], cache oblivious Tiling [55, 54, 17], split-tiling [25] and Tessellation [63], which are mostly compiler transformation techniques. A variety of auto-tuning frameworks [11, 23, 30, 65] have also been presented by using varied hyper-rectangular tiles to exploit data reuse alone. However, redundant computations are involved in these work to resolve the introduced inter-tile dependencies especially when combined with vectorization, which hinder the concurrent execution on different cores.

With the interests in more efficient solutions, Stencil on GPU is also being exploited in the community [26, 65, 21]. Common stencil optimizations on GPU include tiling and loop unrolling [26, 31, 39], however, correct and efficient implementation of these techniques are challenging because careful utilization of limited registers and shared memory is required for complex data dependencies [37]. Since GPU runs low-precision operations (FP16/FP32) highly faster than high precision (FP64), most existing work employ low-precision designs while they are only applied to a limited set of benchmarks and hardly meet the strict tastes of scientific computing [36, 42]. Zhao presents a general stencil framework by exploiting data reuse within a block [66], however, the algorithms are implemented separately on CPUs and GPUs in practice. AN5D achieves high-degree temporal blocking on GPUs, while it only conducts the parameter search with single-precision, and the method is limited to CUDA cores [37].

8 Conclusion

This paper presents Tetris, a disruptive system that democratizes Stencil-driven scientific computing on CPU and GPU of Cloud with polymorphic tiling tetrominoes to tessellate the spatial and temporal dimensions perfectly. Sophisticated Pattern Mapping by register-level tetrominoes, efficient Locality Enhancer by

cache/SMEM-level tetrominoes, and novel concurrent scheduler by memory-level tetrominoes are also first proposed to improve the performance on heterogeneous architecture. Our promising results demonstrate that Tetris can achieve an unprecedented performance improvement compared to the existing state-of-the-arts.

References

[1] AHMAD, Z., CHOWDHURY, R., DAS, R., GANAPATHI, P., GREGORY, A., AND ZHU, Y. Fast stencil computations using fast fourier transforms. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures* (2021), pp. 8–21.

[2] ANDO, K., BALE, R., LI, C., MATSUOKA, S., ONISHI, K., AND TSUBOKURA, M. Digital transformation of droplet/aerosol infection risk assessment realized on" fugaku" for the fight against covid-19. *arXiv preprint arXiv:2110.09769* (2021).

[3] ASANOVIC, K., BODIK, R., CATANZARO, B. C., GEBIS, J. J., HUSBANDS, P., KEUTZER, K., PATTERSON, D. A., PLISHKER, W. L., SHALF, J., WILLIAMS, S. W., ET AL. The landscape of parallel computing research: A view from berkeley.

[4] ASANOVIC, K., BODIK, R., DEMMEL, J., KEAVENY, T., KEUTZER, K., KUBIATOWICZ, J. D., LEE, E. A., MORGAN, N., NECULA, G., PATTERSON, D. A., ET AL. The parallel computing laboratory at uc berkeley: A research agenda based on the berkeley view. *EECS Department, University of California, Berkeley, Tech. Rep* (2008).

[5] BAILEY, D. H. High-precision floating-point arithmetic in scientific computation. *Computing in science & engineering 7*, 3 (2005), 54–61.

[6] BAILEY, D. H., BARRIO, R., AND BORWEIN, J. M. High-precision computation: Mathematical physics and dynamics. *Applied Mathematics and Computation 218*, 20 (2012), 10106–10121.

[7] BANDISHTI, V., PANANILATH, I., AND BONDHUGULA, U. Tiling stencil computations to maximize parallelism. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2012), IEEE, pp. 1–11.

[8] BLUMAN, G. W., AND COLE, J. D. The general similarity solution of the heat equation. *Journal of mathematics and mechanics 18*, 11 (1969), 1025–1042.

[9] BONDHUGULA, U., HARTONO, A., RAMANUJAM, J., AND SADAYAPPAN, P. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2008), pp. 101–113.

[10] CAI, S., WANG, Z., WANG, S., PERDIKARIS, P., AND KARNIADAKIS, G. E. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer 143*, 6 (2021).

[11] CHRISTEN, M., SCHENK, O., AND BURKHART, H. Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In *IPDPS 2011*, IEEE.

[12] CHRISTEN, M., SCHENK, O., AND CUI, Y. Patus for convenient high-performance stencils: evaluation in earthquake simulations. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2012), IEEE, pp. 1–10.

[13] DAKKAK, A., LI, C., XIONG, J., GELADO, I., AND HWU, W.-M. Accelerating reduction and scan using tensor core units. In *Proceedings of the ACM International Conference on Supercomputing* (2019), pp. 46–57.

[14] DATTA, K., MURPHY, M., VOLKOV, V., WILLIAMS, S., CARTER, J., OLIKER, L., PATTERSON, D., SHALF, J., AND YELICK, K. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), IEEE, pp. 1–12.

[15] DENZLER, A., BERA, R., HAJINAZAR, N., SINGH, G., OLIVEIRA, G. F., GÓMEZ-LUNA, J., AND MUTLU, O. Casper: Accelerating stencil computation using near-cache processing. *arXiv preprint arXiv:2112.14216* (2021).

[16] DING, C., AND HE, Y. A ghost cell expansion method for reducing communications in solving pde problems. SC '01, pp. 50–50.

[17] FRIGO, M., AND STRUMPEN, V. Cache oblivious stencil computations. ICS '05, pp. 361–366.

[18] FU, H., HE, C., CHEN, B., YIN, Z., ZHANG, Z., ZHANG, W., ZHANG, T., XUE, W., LIU, W., YIN, W., ET AL. 18.9-pflops nonlinear earthquake simulation on sunway taihulight: enabling depiction of 18-hz and 8-meter scenarios. In *Proceedings*

of the International Conference for High Performance Computing, Networking, Storage and Analysis (2017), pp. 1–12.

[19] FU, H., LIAO, J., DING, N., DUAN, X., GAN, L., LIANG, Y., WANG, X., YANG, J., ZHENG, Y., LIU, W., ET AL. Redesigning cam-se for peta-scale climate modeling performance and ultra-high resolution on sunway taihulight. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), pp. 1–12.

[20] GRANLUND, T. Instruction latencies and throughput for amd and intel x86 processors. In *Technical report*. KTH, 2012.

[21] GROSSER, T., VERDOOLAEGE, S., COHEN, A., AND SADAYAPPAN, P. *The Promises of Hybrid Hexagonal/Classical Tiling for GPU*. PhD thesis, INRIA, 2013.

[22] GUIDE, I. I. Url: https://software. intel. com/sites/-landingpage. *IntrinsicsGuide (access date: 21.10. 2019) 78*.

[23] GYSI, T., GROSSER, T., AND HOEFLER, T. Modesto: Data-centric analytic optimization of complex stencil programs on heterogeneous architectures. In *ICS 2015* (2015), pp. 177–186.

[24] HENRETTY, T., STOCK, K., POUCHET, L.-N., FRANCHETTI, F., RAMANUJAM, J., AND SADAYAPPAN, P. Data layout transformation for stencil computations on short-vector simd architectures. In *International Conference on Compiler Construction* (2011), Springer, pp. 225–245.

[25] HENRETTY, T., VERAS, R., FRANCHETTI, F., POUCHET, L.-N., RAMANUJAM, J., AND SADAYAPPAN, P. A stencil compiler for short-vector simd architectures. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing* (New York, NY, USA, 2013), ICS '13, Association for Computing Machinery, pp. 13–24.

[26] HOLEWINSKI, J., POUCHET, L.-N., AND SADAYAPPAN, P. High-performance code generation for stencil computations on gpu architectures. In *Proceedings of the 26th ACM international conference on Supercomputing* (2012), pp. 311–320.

[27] IRIGOIN, F., AND TRIOLET, R. Supernode partitioning. POPL '88, pp. 319–329.

[28] JIA, W., WANG, H., CHEN, M., LU, D., LIN, L., CAR, R., WEINAN, E., AND ZHANG, L. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In *SC20: International conference for high performance computing, networking, storage and analysis* (2020), IEEE, pp. 1–14.

[29] JIN, G., MELLOR-CRUMMEY, J., AND FOWLER, R. Increasing temporal locality with skewing and recursive blocking. SC '01, pp. 43–43.

[30] KAMIL, S., CHAN, C., OLIKER, L., SHALF, J., AND WILLIAMS, S. An auto-tuning framework for parallel multicore stencil computations. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)* (2010), IEEE, pp. 1–12.

[31] KRISHNAMOORTHY, S., BASKARAN, M., BONDHUGULA, U., RAMANUJAM, J., ROUNTEV, A., AND SADAYAPPAN, P. Effective automatic parallelization of stencil computations. *ACM sigplan notices 42*, 6 (2007), 235–244.

[32] LAM, M. D., ROTHBERG, E. E., AND WOLF, M. E. The cache performance and optimizations of blocked algorithms. ASPLOS IV, pp. 63–74.

[33] LI, K., SHANG, H., ZHANG, Y., LI, S., WU, B., WANG, D., ZHANG, L., LI, F., CHEN, D., AND WEI, Z. Openkmc: a kmc design for hundred-billion-atom simulation using millions of cores on sunway taihulight. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2019), pp. 1–16.

[34] LI, K., YUAN, L., ZHANG, Y., AND YUE, Y. 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* (2021), pp. 1–15.

[35] LI, K., YUAN, L., ZHANG, Y., YUE, Y., AND CAO, H. An efficient vectorization scheme for stencil computation. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2022), IEEE, pp. 650–660.

[36] MARUYAMA, N., AND AOKI, T. Optimizing stencil computations for nvidia kepler gpus. In *Proceedings of the 1st international workshop on high-performance stencil computations, Vienna* (2014), Citeseer, pp. 89–95.

[37] MATSUMURA, K., ZOHOURI, H. R., WAHIB, M., ENDO, T., AND MATSUOKA, S. 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* (2020), pp. 199–211.

[38] MCKELLAR, A. C., AND COFFMAN, JR., E. G. Organizing matrices and matrix operations for paged memory systems. *Commun. ACM 12*, 3 (1969), 153–165.

[39] MENG, J., AND SKADRON, K. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus. In *Proceedings of the 23rd international conference on Supercomputing* (2009), pp. 256–265.

[40] MIT. Stencil computing, 2010. [Online; accessed 29-Nov-2022].

[41] NGUYEN, A., SATISH, N., CHHUGANI, J., KIM, C., AND DUBEY, P. 3.5-d blocking optimization for stencil computations on modern cpus and gpus. SC '10, pp. 1–13.

[42] NGUYEN, A., SATISH, N., CHHUGANI, J., KIM, C., AND DUBEY, P. 3.5-d blocking optimization for stencil computations on modern cpus and gpus. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (2010), IEEE, pp. 1–13.

[43] NVIDIA. Cuda toolkit v11.8.0, 2022. [Online; accessed 29-Nov-2022].

[44] NVIDIA. Solve the world's greatest challenges with supercomputing, 2022. [Online; accessed 29-Nov-2022].

[45] OGBOLE, M. O., OGBOLE, E. A., AND OLAGESIN, A. Cloud systems and applications: A review. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* (2021), 142–149.

[46] ON PHYSICAL MATHEMATICAL ENGINEERING SCIENCE, T. C. Grand challenges: High performance computing and communications. Tech. rep., The FY 1992 US Research and Development Program, 1992.

[47] PISHA, L., AND LIGOWSKI, Ł. Accelerating non-power-of-2 size fourier transforms with gpu tensor cores. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2021), IEEE, pp. 507–516.

[48] RAISSI, M., PERDIKARIS, P., AND KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics 378* (2019), 686–707.

[49] RASTELLO, F., AND DAUXOIS, T. Efficient tiling for an ode discrete integration program: Redundant tasks instead of trapezoidal shaped-tiles. IPDPS '02, pp. 138–.

[50] REN, J., RAJBHANDARI, S., AMINABADI, R. Y., RUWASE, O., YANG, S., ZHANG, M., LI, D., AND HE, Y. {ZeRO-Offload}: Democratizing {Billion-Scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (2021), pp. 551–564.

[51] RIVERA, G., AND TSENG, C.-W. Tiling optimizations for 3d scientific computations. SC '00.

[52] SOH, J., COPELAND, M., PUCA, A., AND HARRIS, M. Microsoft azure and cloud computing. In *Microsoft Azure*. Springer, 2020, pp. 3–20.

[53] SONG, Y., AND LI, Z. New tiling techniques to improve cache temporal locality. PLDI '99, pp. 215–228.

[54] STRZODKA, R., SHAHEEN, M., PAJAK, D., AND SEIDEL, H.-P. Cache oblivious parallelograms in iterative stencil computations. In *Proceedings of the 24th ACM International Conference on Supercomputing* (2010), pp. 49–59.

[55] TANG, Y., CHOWDHURY, R. A., KUSZMAUL, B. C., LUK, C.-K., AND LEISERSON, C. E. The pochoir stencil compiler. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures* (New York, NY, USA, 2011), SPAA '11, Association for Computing Machinery, pp. 117–128.

[56] WOLF, M. E., AND LAM, M. S. A data locality optimizing algorithm. PLDI '91, pp. 30–44.

[57] WOLFE, M. More iteration space tiling. Supercomputing '89, pp. 655–664.

[58] WONNACOTT, D. Achieving scalable locality with time skewing. *Int. J. Parallel Program. 30*, 3 (June 2002), 181–221.

[59] XIAO, J., LI, S., WU, B., ZHANG, H., LI, K., YAO, E., ZHANG, Y., AND TAN, G.

Communication-avoiding for dynamical core of atmospheric general circulation model. In *Proceedings of the 47th International Conference on Parallel Processing* (2018), pp. 1–10.

[60] XU, S., WANG, W., ZHANG, J., JIANG, J., JIN, Z., AND CHI, X. High performance computing algorithms and software for heterogeneous computing. *Journal of Software 32*, 8 (2021), 2365–2376.

[61] YOUNT, C. Vector folding: Improving stencil performance via multi-dimensional simd-vector representation. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems* (2015), IEEE, pp. 865–870.

[62] YOUNT, C., TOBIN, J., BREUER, A., AND DURAN, A. Yask-yet another stencil kernel: A framework for hpc stencil code-generation and tuning. In *2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)* (2016), IEEE, pp. 30–39.

[63] YUAN, L., HUANG, S., ZHANG, Y., AND CAO, H. Tessellating star stencils. In *Proceedings of the 48th International Conference on Parallel Processing* (2019), pp. 1–10.

[64] YUAN, L., ZHANG, Y., GUO, P., AND HUANG, S. Tessellating stencils. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2017), SC '17, Association for Computing Machinery.

[65] ZHANG, Y., AND MUELLER, F. Auto-generation and auto-tuning of 3d stencil codes on gpu clusters. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization* (2012), pp. 155–164.

[66] ZHAO, T., BASU, P., WILLIAMS, S., HALL, M., AND JOHANSEN, H. Exploiting reuse and vectorization in blocked stencil computations on cpus and gpus. In *SC'19* (2019), pp. 1–44.