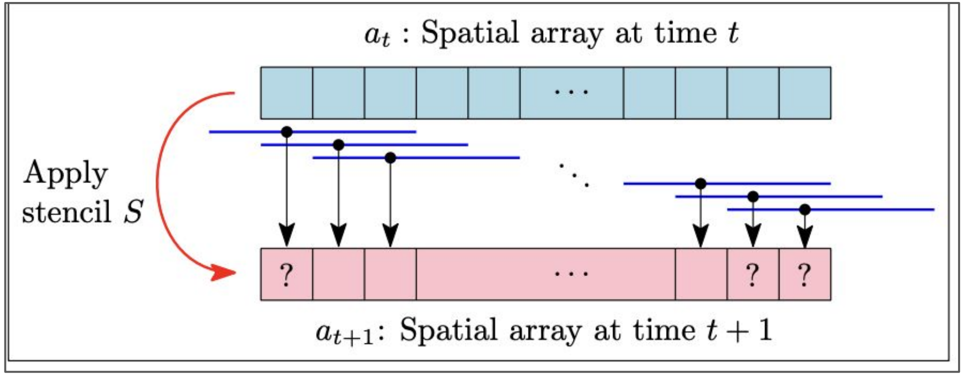
Fast Stencil Computations using Fast Fourier Transforms

Zafar Ahmad, Rezaul Chowdhury, Rathish Das, Pramod Ganapathi, Aaron Gregory, Yimin Zhu

Stencil Computations

- Stencil: pattern used to compute the value of a cell using neighboring cells from previous time steps
- We usually want to perform a stencil computation over some time steps T
- Lots of applications in scientific computing [fluid dynamics, electromagnetics, image processing, etc.]
- Current stencil compilers that use a linear stencil have runtimes bounded by **Θ(NT)**



Types of Boundaries

- Periodic Boundaries: The space grid wraps around itself - e.g. top cells are adjacent to bottom cells, right boundary cells are adjacent to left boundary cells

- Aperiodic Boundaries: Cells on the boundaries are calculated via some other method (aka not the stencil)

Other Methods

- Looping Algorithms
- Tiled Looping Algorithms
- Recursive Divide and Conquer Algorithms [trapezoidal decomposition alg.]
- Krylov Subspace Methods -> Iterative method [applies to certain PDEs]
  - Runtime vs accuracy
  - Preconditioner

Problems with other methods

- Manual analysis
- Overspecialization
- Inexact solutions
- Nonoptimal computation complexity

Problem Statement

Problem Statement

- Suppose $a_t[0, \ldots, N-1]$ represents the N elements in our 1-dimensional space grid at time T
- Suppose we have a **linear stencil S** that is applied to every cell at every time step: this stencil can be represented as an NxN matrix
- Thus a time step looks like this: $a_{t+1}[i] = \sum_j S[i,j] a_t[j]$.
- Because a stencil bases its computation off of relative indices, our stencil can be represented as a composition of right-shift matrices -> it is also **circulant**

$$\underbrace{\begin{bmatrix} s_0 & s_{N-1} & \cdots & s_2 & s_1 \\ s_1 & s_0 & s_{N-1} & & s_2 \\ \vdots & s_1 & s_0 & \ddots & \vdots \\ s_{N-2} & & \ddots & \ddots & s_{N-1} \\ s_{N-1} & s_{N-2} & \cdots & s_1 & s_0 \end{bmatrix}}_{S}$$

Periodic FFT Stenciling

$$a_T = S^T a_0$$

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

$$\mathcal{F}S^T\mathcal{F}^{-1} = \mathcal{F}S\mathcal{F}^{-1}\mathcal{F}S\mathcal{F}^{-1}\cdots\mathcal{F}S\mathcal{F}^{-1} = (\mathcal{F}S\mathcal{F}^{-1})^T$$

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

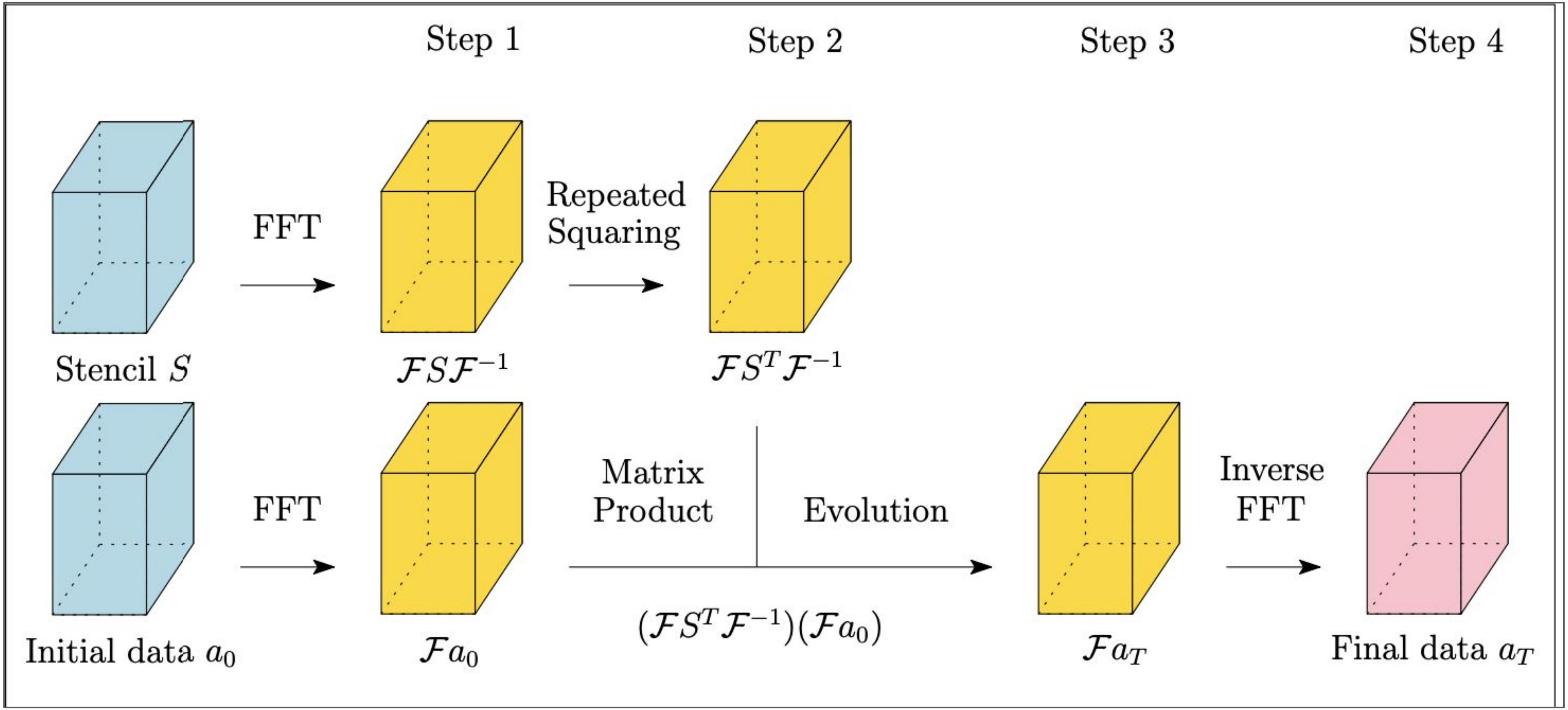
$$\longrightarrow \quad a_T = \mathcal{F}^{-1} \left(\mathcal{F} S \mathcal{F}^{-1}\right)^T \mathcal{F} a_0.$$

Because S is circulant, it is diagonalizable by FFT

$$\longrightarrow \quad \mathcal{F} S \mathcal{F}^{-1} = \Lambda$$

$$\mathcal{F} a_0 = x$$

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



Step 1
Step 2
Step 3
Step 4
FFT
Repeated Squaring
Stencil S
$\mathcal{F}S\mathcal{F}^{-1}$
$\mathcal{F}S^T\mathcal{F}^{-1}$
FFT
Matrix Product
Evolution
Inverse FFT
$(\mathcal{F}S^T\mathcal{F}^{-1})(\mathcal{F}a_0)$
Initial data a_0
$\mathcal{F}a_0$
$\mathcal{F}a_T$
Final data a_T

Analysis

FFT

$$\mathcal{F} S \mathcal{F}^{-1} \qquad \mathcal{F} a_0$$

- Since we only need to compute the FFT of the first column of S, this takes O(Nlog(N))
- The same amount of work occurs in $\mathcal{F} a_0$
- Span is log(N)loglog(N)

Repeated Squaring

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

- Must calculate squares for each binary one, and then multiply
- For a number T, there are log(T) bits
- Each power of 2 can be calculated in parallel each with O(N) work, and then merged in O(log(T)) time steps
- Work: O(Nlog(T)) Span: O(log(N) + log(T))

Element-wise Product

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

- Because $(\mathcal{F} S^T \mathcal{F}^{-1})$ is diagonal, we can compute this in parallel (each entry in $(\mathcal{F} a_0)$ is multiplied by a diagonal value)

- Work is thus O(N). Span is O(log(N)) (because of binary forking model)

Inverse FFT

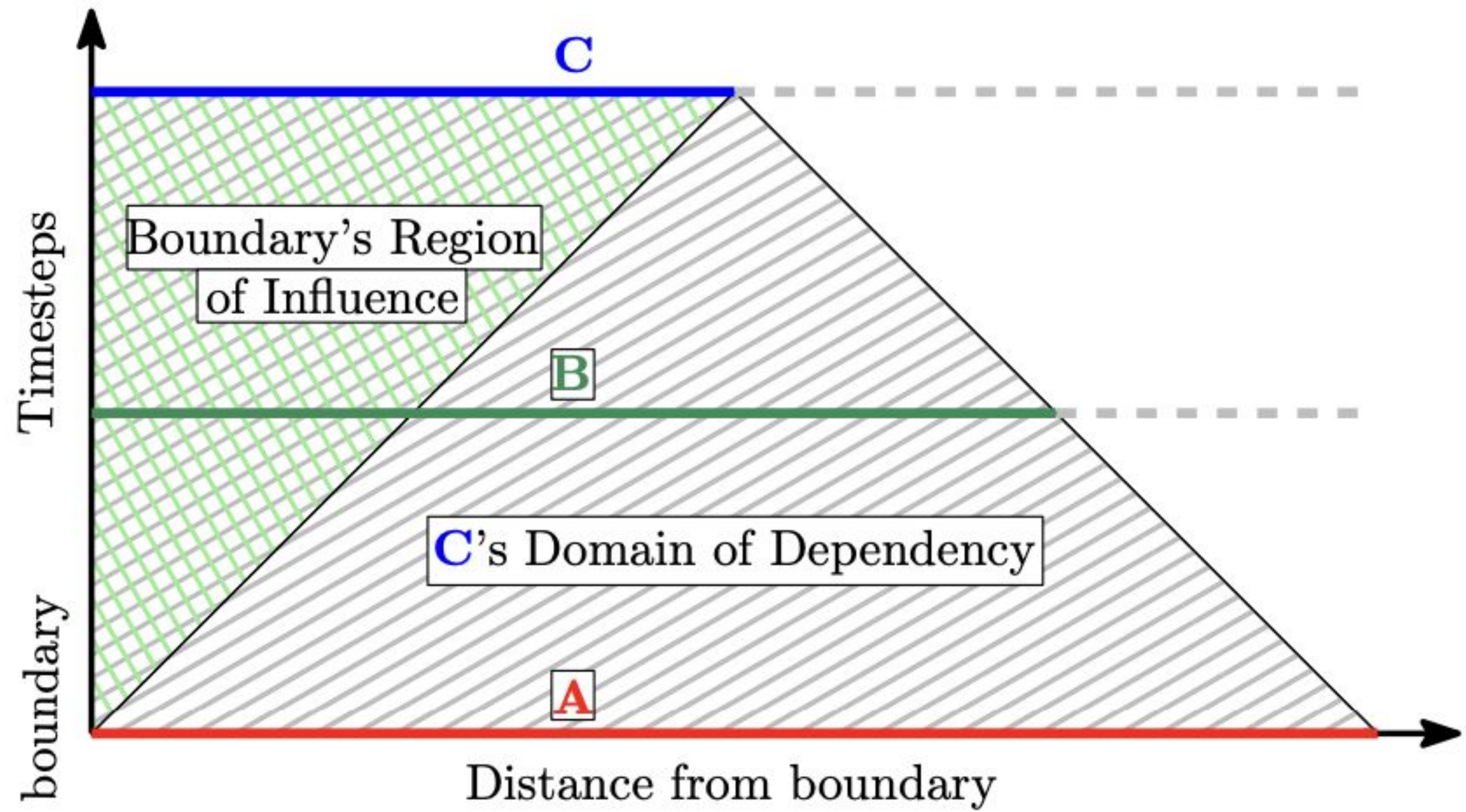
Applying the inverse FFT is the same work and span as applying an FFT

Work: $O(N\log(N))$
Span: $O(\log(N)\log\log(N))$

Aperiodic StencilFFT

Boundary’s Region of Influence

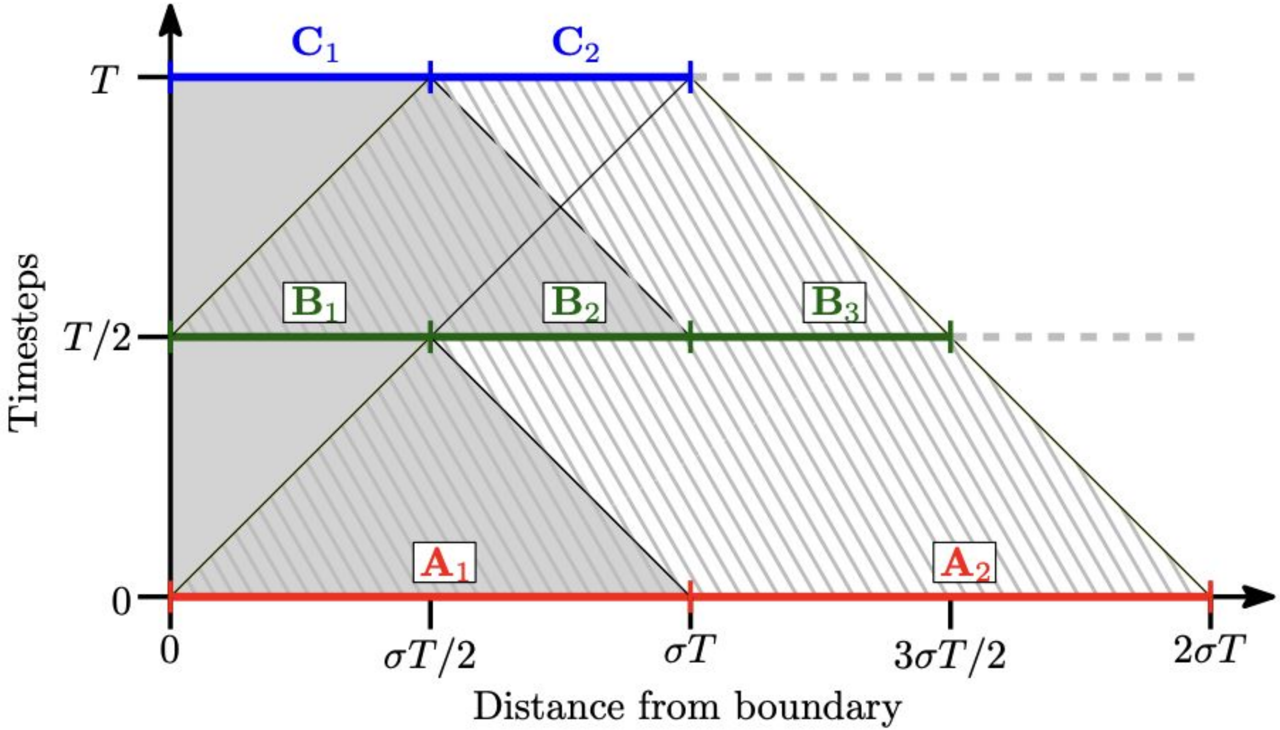
- Set of nodes that depend on boundary conditions after T timesteps
- If we have a stencil with radius σ, then the region of influence will include nodes within distance σT from the boundary
- Any nodes not in the region of influence can be calculated for T timesteps, if it is not in the region of influence


C
Boundary's Region
of Influence
B
C's Domain of Dependency
A
Timesteps
boundary
Distance from boundary

Domain of Dependence

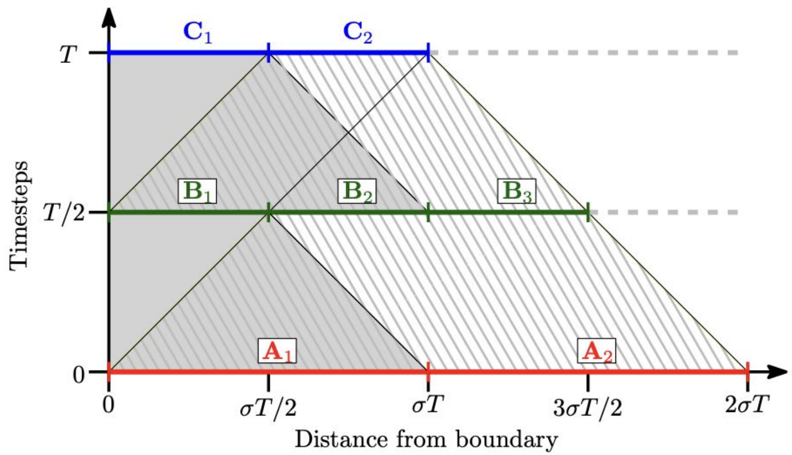
- Nodes whose values we need in order to calculate the region of influence after T time steps
- (The final value is dependent on these outer values)

Recursive Boundary Routine



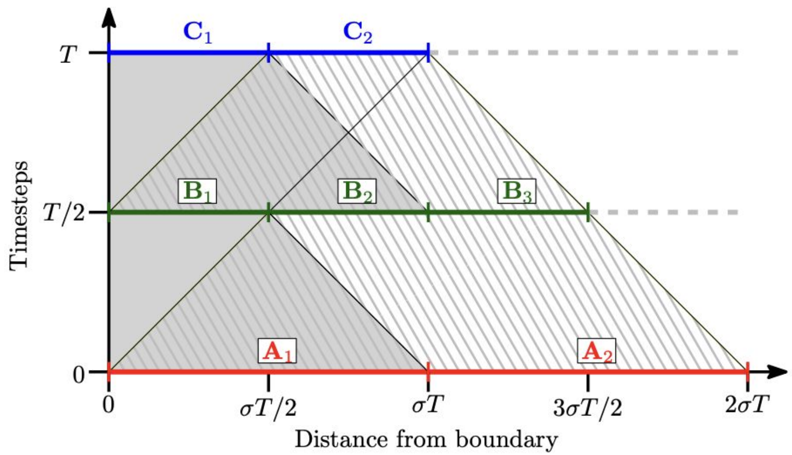
Recursive Boundary Routine

- A_1 and A_2 are necessary to calculate B_2 & B_3.
- B_1 is in the region of influence and thus requires boundary condition calculations. This is recursively solved using A_1
- After solving for B_1, B_2, and B_3, its as if we are starting from T = 0, and no value is in the region of influence



Recursive Boundary Routine

- C_1 requires B_1 and B_2
- C_2 requires B_1, B_2, and B_3



Analysis

Work: $\Theta\left(bT\log(bT)\log T + N\log N\right)$

Span: $\Theta\left(T\log b + \log N\log\log N)\right)$

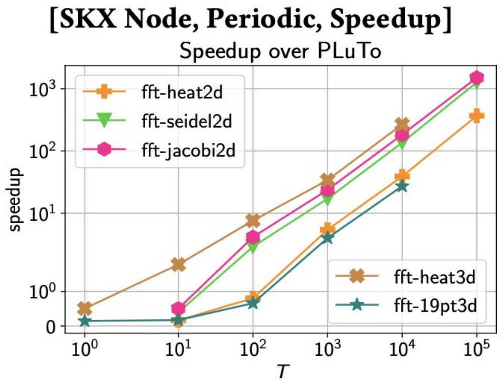
Experimental Results

Machines

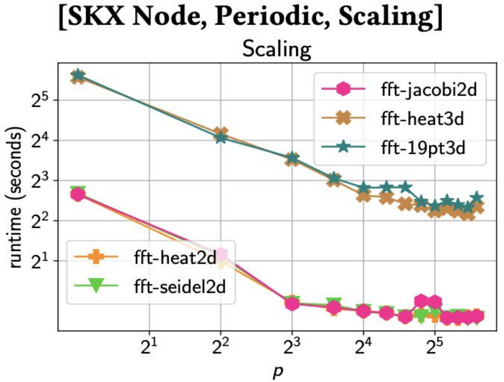
KNL	Cores	68 cores per socket, 1 socket (total: 68 threads)
	Cache sizes	L1 32 KB, L2 1 MB, L3 16 GB (shared)
	Memory	96 GB DDR RAM
SKX	Cores	24 cores per socket, 2 sockets (total: 48 cores)
	Cache sizes	L1 32 KB, L2 1 MB, L3 33 MB
	Memory	144GB /tmp partition on a 200GB SSD
Compiler		Intel C++ Compiler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

Periodic Stenciling Speedup

- As N is kept constant, and T increases, the speedup between the new stenciling algorithm and PluTo's implementation increases
- This follows from the theoretical bound factor of O(T/log(T))



(i)



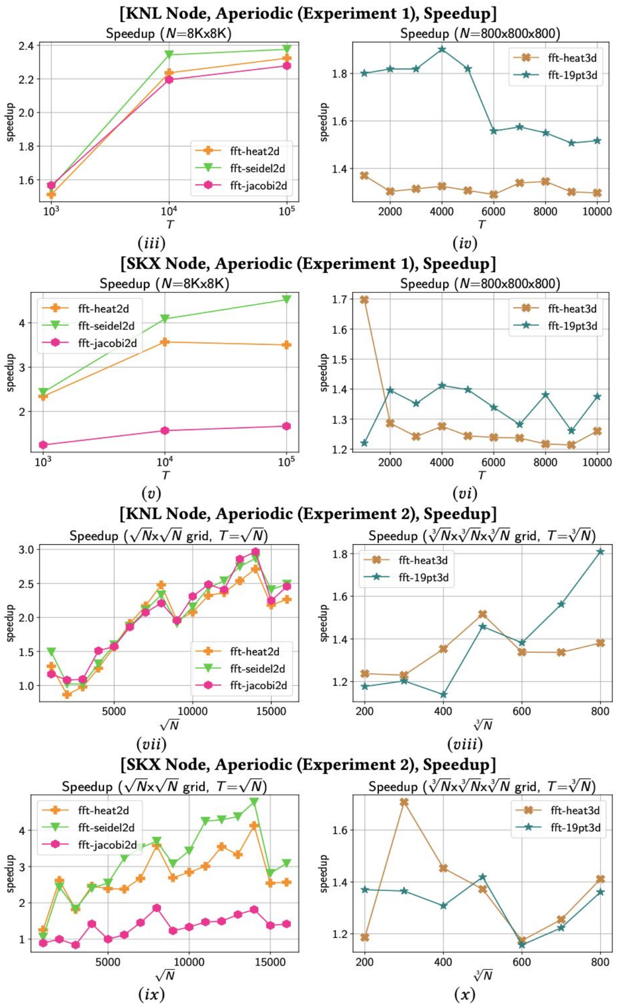
(ii)

Aperiodic Stenciling Speedup

- Two types of experiments
  - N is constant, T is varied
  - Grid was set to N^(1/d) * N^(1/d) … * N^(1/d) and T = N^(1/d) for dimension d & N was varied

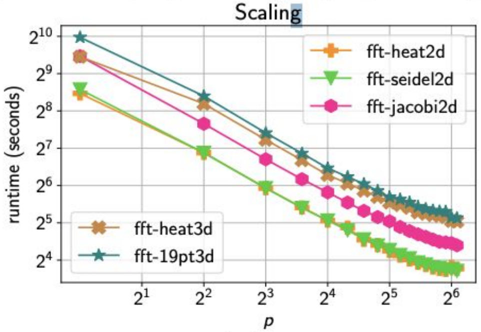
$$\Theta\left(N^{1/d} / \left(\log\left(TN^{1-1/d}\right)\log T\right)\right)$$

Drops due to new kernel base size



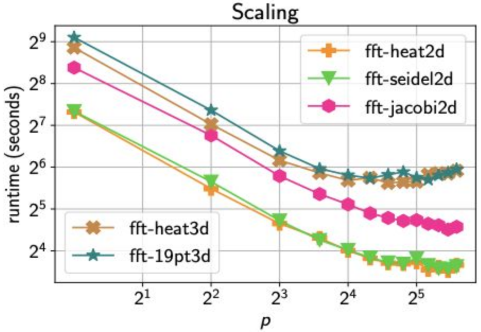
Aperiodic Stenciling Scaling

[KNL Node, Aperiodic (Exp. 2), Scaling]



(xi)

[SKX Node, Aperiodic (Exp. 2), Scaling]



(xii)

Strengths, Weaknesses, Future Work

Strengths:

- Smart integration of periodic stenciling into situations w/ aperiodic boundary conditions
- Intuitive analyses for theoretic bounds

Weaknesses:

- Weird metrics (T=N^(1/d) [is this some sort of standard?]
- Not much motivation behind using FFT besides properties of matrices

Future Work:

- Improved stenciling for inhomogeneous / nonlinear stencils
- Focusing on ways to reduce memory bottleneck?

Discussion Questions

- How commonly is this algorithm used for basic stencil computations
- Are there approximation algorithms / iterative solvers that trade accuracy with speed, especially with interesting recursive solutions like what we just saw?