High Performance FFT Code Generation through MLIR Linalg Dialect and Micro-kernel

Yifei He, Stefano Markidis

KTH Royal Institute of Technology, Sweden

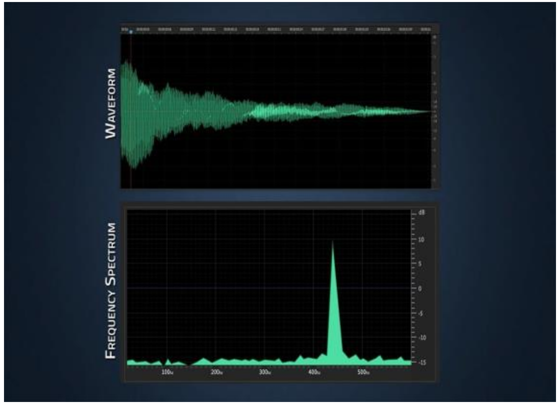
Outline

- Motivation
- Background
- Methdology
- Insights
- Future Work

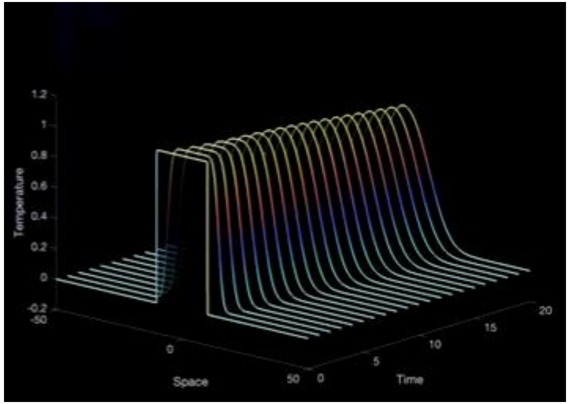
Motivation: Importance of Fast Fourier Transform

- **Applications**

Signal processing



Partial Differential Equations(PDE)



- **Libraries for FFT:**

FFTW SPIRAL heFFTe Intel oneAPI Math Kernel Library CUFFT

Background: FFT Algorithm in Matrix-Formalism

$\mathcal{O}(n^2)$

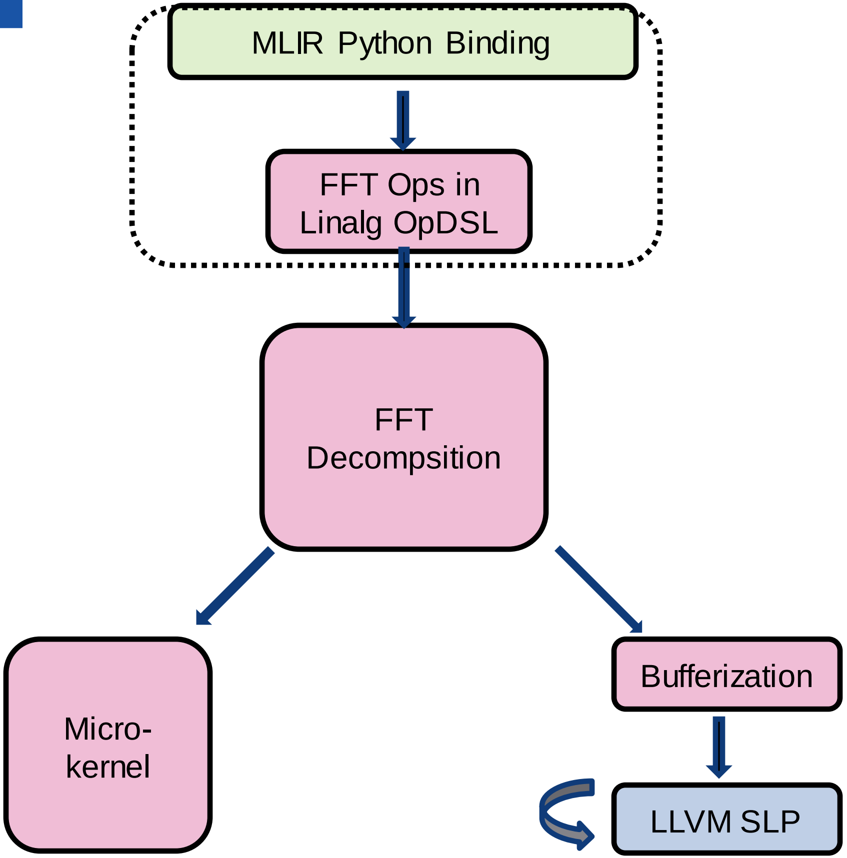
$$DFT_{N_{m,n}} = (\omega_N)^{mn}, \quad \text{where} \quad \omega_N = \exp(-2\pi i/N) \quad \text{for} \quad 0 \leq m, n < N.$$

$$\mathrm{DFT_N} = (\mathrm{DFT_K} \otimes \mathrm{I_M})\, \mathrm{D^N_M} (\mathrm{I_K} \otimes \mathrm{DFT_M})\, \Pi^{\mathrm{N}}_{\mathrm{K}} \quad \text{with} \quad N = MK$$

$\mathcal{O}(n \log n)$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{bmatrix}}_{\mathrm{DFT_2} \otimes \mathrm{I_2}} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -i \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix}}_{\mathrm{I_2} \otimes \mathrm{DFT_2}} \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix}$$

Implementation: Compilation Pipeline



Complex Arithmetic not supported well in MLIR/LLVM

Python friendly
- Generate MLIR from Python Binding
- Pass manager in Python

JIT:
- Input/output as Python array

AOT:
- Implemented as a C library
- Input/output as C buffer

Implementation Linalg: Utilize Sparsity in FFT Computation

FFTc DSL Pattern	Sparse Fusion	Bufferization
$Y = (A_m \otimes I_n) \cdot X$	FusedMKIV(A, n, X)	$\text{for}(i = 0;\ i < n;\ i++)$ $\text{Y}[i : n : i + m * n - n] =$ $\text{A}*(\text{X}[i : n : i + m * n - n])$
$Y = (I_m \otimes A_n) \cdot X$	FusedIKMV(A, n, X)	$\text{for}(i = 0;\ i < m;\ i++)$ $\text{Y}[i * n : 1 : i * n + n - 1] =$ $\text{A}*(\text{X}[i * n : 1 : i * n + n - 1])$
$(\Pi_m^{mn} \otimes I_k) \cdot X$	FusedPKIV(m, mn, k, X)	$\text{for}(i = 0;\ i < m;\ i++)$ $\text{for}(j = 0;\ j < n;\ j++)$ $\text{Y}[k * (i + m * j) : 1 : k * (i + m * j)] =$ $\text{X}[k * (n * i + j) : 1 : k * (n * i + j)]$
$D_m^n \cdot X$	Mul(TwiddleCoe, X)	$\text{for}(i = 0;\ i < m;\ i++)$ $\text{Y}[i] = D_m^n[i] * \text{X}[i])$
$\Pi_m^{mn} \cdot X$	Permute(m, mn, X)	$\text{for}(i = 0;\ i < m;\ i++)$ $\text{for}(j = 0;\ j < n;\ j++)$ $\text{Y}[i + m * j : 1 : i + m * j] =$ $\text{A}*(\text{X}[n * i + j : 1 : n * i + j])$

Source: He, Yifei, Artur Podobas, and Stefano Markidis. "Leveraging MLIR for Loop Vectorization and GPU Porting of FFT Libraries." *arXiv e-prints* (2023): arXiv-2308.

Implementation Linalg: FFT Operations in MLIR Linalg OpDSL

$$Y = (A_m \otimes I_n) \cdot X$$

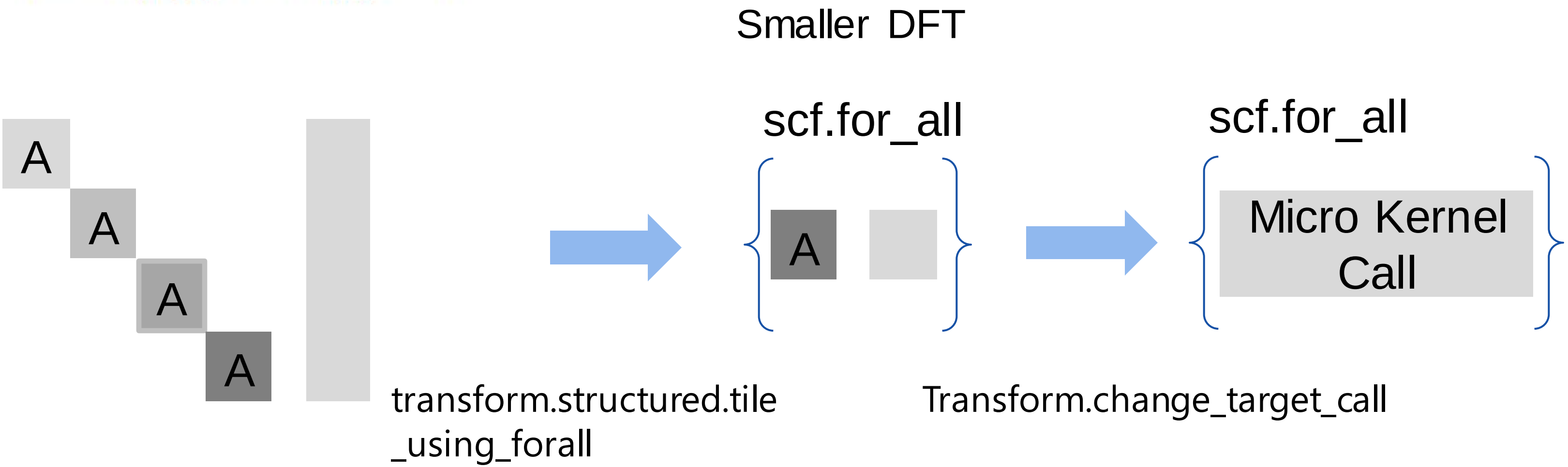
```
@linalg_structured_op
def AkI_x(
    K=TensorDef(T1, S.KN, index_dims=[D.kn]),
    A=TensorDef(T1, S.M, S.N),
    B=TensorDef(T1, S.N * S.KN),
    C=TensorDef(U, S.M * S.KN, output=True),
    strides=IndexAttrDef(S.SM, default=[1]),
    cast=TypeFnAttrDef(default=TypeFn.cast_signed),
):
    domain(D.kn, D.m, D.n)
    C[D.kn + D.m * S.SM] += cast(U, A[D.m, D.n]) *
     cast(U, B[D.kn + D.n * S.SM])
```

FFT patterns not supported well in OpDSL:

- Work around: Redundant tensor to specify iteration domain dimension

Implementation Linalg: Map to micro-kernel with transform dialect

$$Y = (I_m \otimes A_n) \cdot X$$

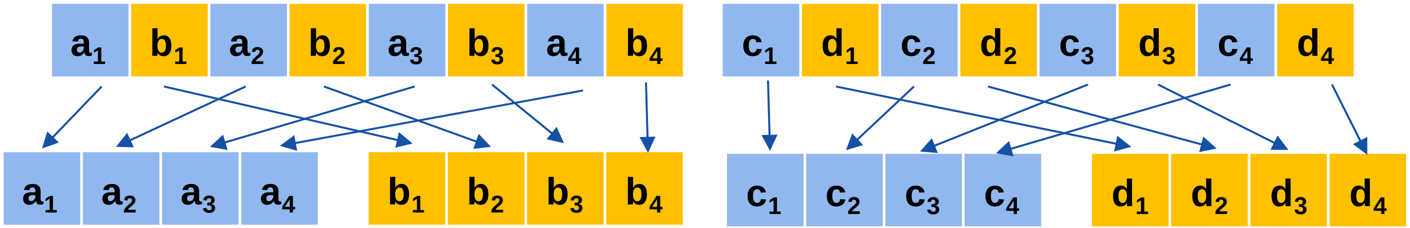


Implementation Micro-kernel: Data layout

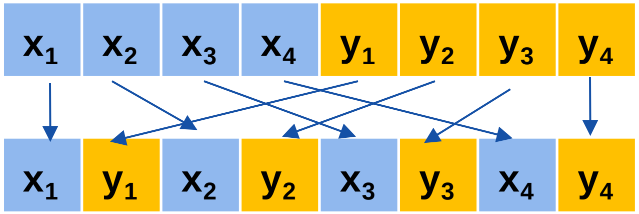
SIMD friendly Data Layout for Complex Arithmetic

$$(a+bi) * (c+di) = ac-bd + (ad+bc)i$$

Interleaved



$$(a+bi) * (c+di) = ac-bd + (ad+bc)i$$



Source: Popovici, Doru T., Franz Franchetti, and Tze Meng Low. "Mixed data layout kernels for vectorized complex arithmetic." *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017.

Implementation Micro-kernel: Data layout

SIMD friendly Data Layout for Complex Arithmetic

$$(a+bi) * (c+ di) = ac-bd + (ad+bc)i$$

Block-interleaved

$$(a+bi) * (c+ di) = ac-bd + (ad+bc)i$$

Source: Popovici, Doru T., Franz Franchetti, and Tze Meng Low. "Mixed data layout kernels for vectorized complex arithmetic." *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017.

Implementation Micro-kernel: Other optimizations

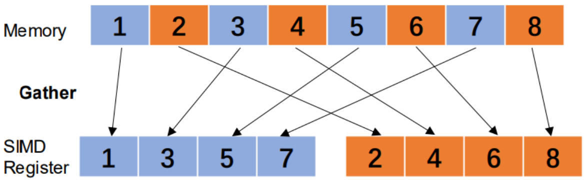
- Memory access optimization for strided permute
  - Multiple iterations of in-register shuffle to implement blocked & strided memory access pattern
  - 10x speed up compared with gather/scatter
- Loop unroll to enable vectorization for small loop trip count
  - Require extra shuffle, can not be done by auto-vectorizer

- Software prefetching

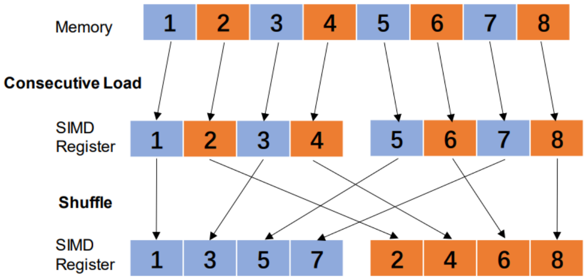
- Pre-computed constants
  - DFT matrix, Twiddle factor

Implementation Code Generation: Auto Vectorization on Complex Array with LLVM SLP Vectorizer

Interleaved memory access optimization for complex array



(a) Directly Load Complex Data Using Gather Instructions



(b) Optimized Interleaved Memory Access

Source: He, Yifei, Artur Podobas, and Stefano Markidis. "Leveraging MLIR for Loop Vectorization and GPU Porting of FFT Libraries." *arXiv e-prints* (2023): arXiv-2308.

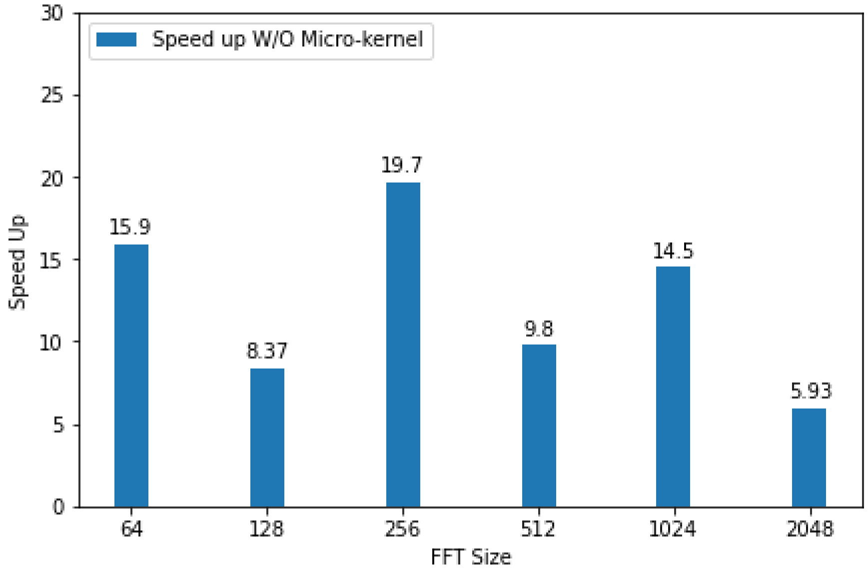
Our Contribution:

- **FFT Representation & Transformation in MLIR Python binding and OpDSL**
  - FFT-specific ops in Linalg
  - FFT decomposition to smaller size
    - Cache friendly
    - Reduce computation complexity
    - Change inner most kernel to micro-kernel call
- **Optimize complex arithmetic in micro-kernel:**
  - Complex values not supported well in MLIR/LLVM
  - Complex arithmetic specific optimization not available in general purpose auto-vectorizer
    - SIMD friendly data layout
    - Memory access optimization

Future Work

- Overhead in buffer allocation & function call between MLIR&C
- Constant propagation
- Currently C++ intrinsics, better register allocation & instruction scheduling with assembly
- Enable complex value vectorization in MLIR
- Enable complex value vectorization in MLIR

Initial results (In progress)



Thanks!

Q&A