A Domain-specific Compilation Framework for High-performance Fast Fourier Transform

Yifei He Artur Podobas Stefano Markidis

Motivation

Discrete Fourier Transforms (DFT) and their efficient formulations, called Fast Fourier Transforms (FFT), are a critical building block for efficient and high-performance data analysis and scientific computing.

Existing FFT libraries such as the Fastest Fourier Transform in the West (`FFTW`) library, was first designed and implemented with compiler technologies, nowadays outdated and support only multicore CPUs and not GPUs:

- **Lack of portability over heterogeneous hardware**
- **Cannot utilize the evolving compiler community**
- **Emit C level code, lack of control on low level compilation**

Frontend: The FFTc DSL

The Cooley-Tukey general-radix decimation-in-time algorithm for an input of size N can be written as:

$$\mathrm{DFT}_N = (\mathrm{DFT}_K \otimes \mathrm{I}_M)\, \mathrm{D}_M^N (\mathrm{I}_K \otimes \mathrm{DFT}_M)\, \Pi_K^N \quad \text{with} \quad N = MK, \tag{1}$$

where Π_K^N is a stride permute operator and D_M^N is a diagonal matrix of *twiddle* factors. DFT for an input of size four, in matrix formulation:

$$\mathrm{DFT}_4 = \underbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix}}_{\mathrm{DFT}_m \otimes \mathrm{I}_n} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -i \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix}}_{\mathrm{I}_2 \otimes \mathrm{DFT}_2} \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix}, \tag{2}$$

An example source code of our DSL is shown in Listing 1. It's designed as a declarative representation of FFT tensor Algorithm. Here, we propose to keep inputs as similar to abstract mathematical expressions as possible, such as Eq. (1).

```
var InputReal <4, 1> = [[1], [2], [3], [4]];
var InputImg  <4, 1> = [[1], [2], [3], [4]];
var InputComplex = createComplex(InputReal, InputImg);
var result =  (DFT(2) ⊗ I(2)) · twiddle(4,2) · (I(2) ⊗ DFT(2)) ·
  Permute(4,2) · InputComplex;
```

Listing 1: DSL FFT language

FFT Dialect: Abstractions in MLIR

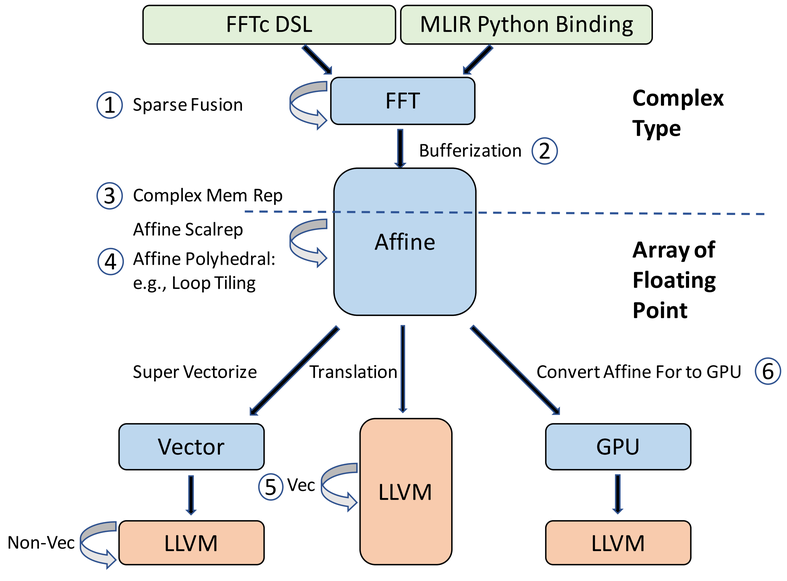
The FFT dialect wraps the operations, attributes, and types to represent the FFT formula.

Table: From FFTc DSL to FFT Dialect MLIR

FFTc DSL	FFT Dialect
createComplex(A, B)	fft.createCT(a,b)
A · B	fft.matmul a, b :
A ⊗ B	fft.kroneckerproduct a, b
twiddle (a,b)	fft.twiddle (a , b)
I(size)	fft.identity (a)
DFT(size)	fft.dft(a)
Permute (a ,b)	fft.Permute(a, b)

The FFT dialect carries high-level information about the FFT computation. We perform the Sparse Fusion Transform (SFT) that performs the tensor products using sparse matrix formats.

FFT Compilation Pipeline

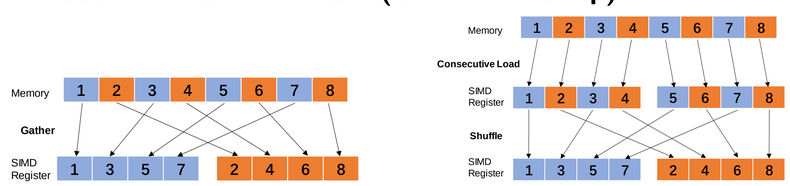


A progressive lowering compilation pipeline, which consists of high-level domain-specific optimizations in MLIR and target-specific transformations in MLIR and LLVM.

Vectorization

We explore three vectorizers in MLIR and LLVM:

- **MLIR Super-vectorize on Affine loops**
- **LLVM SLP Vectorizer (Innermost Loop)**
- **LLVM VPlan Vectorizer (Outermost Loop)**

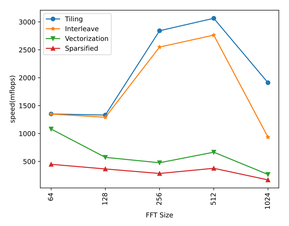


(a) Directly Load Complex Data Using Gather Instructions

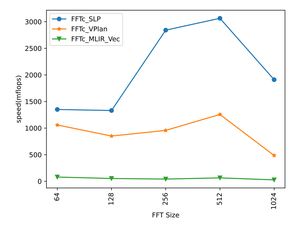
(b) Optimized Interleaved Memory Access

Figure: Pack Complex Data into SIMD Registers during Auto-vectorization

Results



(a) Different and combined optimizations on CPU.



(b) Different vectorizers approaches on CPU.

Future Work