FFTc: An MLIR Dialect for Developing HPC Fast Fourier Transform Libraries

Yifei He, Artur Podobas,
Måns I. Andersson, Stefano Markidis

KTH Royal Institute of Technology
Outline

• Motivation
• Methodology
• Evaluation
• Conclusion & Future work
Motivation
Motivation: Importance of FFT

- Applications

- Libraries for FFT:

  FFTE  FFTPACK  FFTW  CUFFT
The Problems with FFT libraries like FFTW

• Lack of support for modern hardware
  – Newly introduced SIMD/tensor instructions in CPU, GPU, etc

• Lack of portability over heterogeneous hardware
  – Different code generation routines for different backends, cost is high

• Cannot utilize the evolving compiler community
  – MLIR/LLVM is more adaptive to search/learn based methods

• Emit C code, lack of control on low level compilation
FFT Algorithm in matrix-formalism

$O(n^2)$

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

$DFT_N = (DFT_K \otimes I_M) \, D_M^N(I_K \otimes DFT_M) \, I_K^N$ with $N = MK$.

$O(n \log n)$

$$
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -i & -1 & i \\
1 & -1 & 1 & -1 \\
1 & i & -1 & -i
\end{bmatrix} = 
\text{DFT}_2 \otimes I_2 
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 \\
1 & -1 & 1 & 1 \\
1 & -i & 1 & i
\end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = I_2 \otimes \text{DFT}_2
$$
Methodology
**FFTc**: A Domain Specific Compilation for Automatic Generation of FFT Algorithms
**FFTc language:** Declarative representation of FFT tensor Algorithm

\[ \text{DFT}_4 = (\text{DFT}_2 \otimes I_2) D^2_4 (I_2 \otimes \text{DFT}_2) \Pi^2_4 \]

Fourier transform \hspace{2cm} \text{Diagonal matrix (twiddles)}

Kronecker product \hspace{2cm} \text{Identity} \hspace{2cm} \text{Permutation}

```plaintext
1 var InputReal <4, 1> = [[1], [2], [3], [4]]; 
2 var InputImg <4, 1> = [[1], [2], [3], [4]]; 
3 var InputComplex = createComplex(InputReal, InputImg); 
4 var result = (DFT(2) \otimes I(2)) \cdot twiddle(4,2) \cdot (I(2) \otimes \text{DFT}(2)) \cdot \text{Permute}(4,2) \cdot \text{InputComplex}; 
```
FFT Dialect (IR): Operations in FFT Dialect

| 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) |

```
var result =
(DFT(2) ⊗ I(2))
.
.
twiddle(4,2).
.
(I(2) ⊗ I(2))
.
DFT(2))
.
Permute(4,2).
.
InputComplex;

%5 = fft.arithmetic(2.000000e+00 : f64)
%6 = "fft.dft"(%5) : (f64) -> tensor<xcomplex<f64>>
%7 = "fft.arithmetic(2.000000e+00 : f64)
%8 = "fft.identity"(%7) : (f64) -> tensor<xcomplex<f64>>
%9 = fft.kroneckerproduct %6, %8 : tensor<xcomplex<f64>>
%10 = fft.arithmetic(2.000000e+00 : f64)
%11 = fft.arithmetic(2.000000e+00 : f64)
%12 = "fft.twiddle"(%10, %11) : (f64, f64) -> tensor<xcomplex<f64>>
%13 = fft.arithmetic(2.000000e+00 : f64)
%14 = "fft.identity"(%13) : (f64) -> tensor<xcomplex<f64>>
%15 = fft.arithmetic(2.000000e+00 : f64)
%16 = "fft.dft"(%15) : (f64) -> tensor<xcomplex<f64>>
%17 = fft.kroneckerproduct %14, %16 : tensor<xcomplex<f64>>
%18 = fft.arithmetic(2.000000e+00 : f64)
%19 = fft.arithmetic(2.000000e+00 : f64)
%20 = "fft.permute"(%18, %19) : (f64, f64) -> tensor<xcomplex<f64>>
%21 = fft.matmul %20, %4 : tensor<xcomplex<f64>>
%22 = fft.matmul %17, %21 : tensor<xcomplex<f64>>
%23 = fft.matmul %12, %22 : tensor<xcomplex<f64>>
%24 = fft.matmul %9, %23 : tensor<xcomplex<f64>>
```
Progressive Lowering To Affine Dialect

From:

```plaintext
%10 = fft.matmul %9, %3 : (tensor<4x4xcomplex<f64>>,
tensor<4x1xcomplex<f64>>) ->
tensor<4x1xcomplex<f64>>
```

To:

```plaintext
affine.for %arg0 = 0 to 4 {
    affine.for %arg1 = 0 to 1 {
        affine.for %arg2 = 0 to 4 {
            %18 = affine.load %9[%arg0, %arg2] :
            memref<4x4xcomplex<f64>>
            %19 = affine.load %3[%arg2, %arg1] :
            memref<4x1xcomplex<f64>>
            %20 = complex.mul %18, %19 : complex<f64>
        }
    }
}
```
Different Code Generation Modes

Ahead-Of-Time Compilation

**Pros**: Get rid of compilation time  
**Cons**: Fixed FFT size for now

VS

Just-In-Time Compilation

**Pros**: Dynamic FFT size  
**Cons**: Long compilation time
Evaluation
Performance Evaluation

Benchmark:
FFT from input size 32 to 128
Double complex input data
Single thread
Ahead-of-Time compilation mode

Evaluation:
Run for 1000 times, calculate standard deviation for 30 rounds

Hardware:
Dual-socket Intel Xeon Gold 6132 CPU, 192 GB of RAM
Performance Evaluation

**FFT Size 32**
Compile & Run: 6.8903s

**Frontend:** 0.0%

**MLIR Compilation:** 90.4%

**LLVM Middle-end optimization & Code Generation & Run:** 8.9%

**Execution Time Report in JIT Mode**

| Wall Time / Seconds | Name                                      |
|---------------------|-------------------------------------------|
| 0.0034 (0.0%)       | Parser & MLIRGen                          |
| 0.0003 (0.0%)       | Inliner                                   |
| 0.0000 (0.0%)       | (A) CallGraph                             |
| 0.0000 (0.0%)       | ‘builtin.func’ Pipeline                   |
| 0.0002 (0.0%)       | Canonicalizer                             |
| 6.2268 (90.4%)      | ‘builtin.func’ Pipeline                   |
| 0.0001 (0.0%)       | (anonymous)::ShapeInference               |
| 0.0001 (0.0%)       | Canonicalizer                             |
| 0.0000 (0.0%)       | CSE                                       |
| 0.0000 (0.0%)       | (A) DominancelInfo                       |
| 0.0116 (0.2%)       | (anonymous)::AffineToLLVMLoweringPass    |
| 0.0226 (0.4%)       | Canonicalizer                             |
| 0.0014 (0.0%)       | CSE                                       |
| 0.0000 (0.0%)       | (A) DominancelInfo                       |
| 0.6238 (9.1%)       | AffineLoopFusion                          |
| 5.5622 (80.7%)      | AffineScalarReplacement                   |
| 0.0000 (0.0%)       | (A) PostDominancelInfo                   |
| 0.0000 (0.0%)       | (A) DominancelInfo                       |
| 0.0009 (0.0%)       | AffineLoopInvariantCodeMotion             |
| 0.0384 (0.6%)       | (anonymous)::FFTToLLVMLoweringPass       |
| 0.0000 (0.0%)       | output                                    |
| 0.6154 (8.9%)       | Jit                                       |
| 0.0057 (0.1%)       | Rest                                      |
| 6.8903 (100%)       | Total                                     |
Performance Evaluation

- **O2**
  - Inliner, Canonicalizer, CSE
  - Affine: LoopFusion, LoopInvariantCodeMotion
  - LLVM O3 passes

- **O3**
  - MLIR O2 passes
  - Affine: ScalarReplacement
  - LLVM O3 passes
Performance Evaluation

• Reasons contribute to the performance gap with FFTW
  – The FFTs are computed through dense matrix-matrix multiplication
  – Not fully optimized MLIR/LLVM compilation flow
  – No automatic FFT decomposition planner yet
Conclusion & Future Work
Conclusion

• Tensor-based FFT DSL and FFT Dialect in MLIR
  – DSL: Declarative representation of FFT tensor algorithm
  – FFT Dialect: Operations in FFT dialect to represent FFT algorithm

• Code generation pipeline through MLIR and LLVM infrastructure
  – Progressive lowering in MLIR for optimization & transformation at multiple abstraction level
  – Invoke LLVM JIT compilation for lower optimization on LLVM IR & code-generation
Future Work

• Fully Optimized Compilation:
  – FFT formula rewriting (decomposition): Pattern matching & Re-writing in MLIR
  – Loop tiling, vectorization

• Support various hardware backends:
  – CPU tensor unit, GPU, FPGA, etc

• Reduce Compilation Time
  – Multi-threading compilation & remove unnecessary MLIR passes

• Dynamic FFT Size at Compilation Time
  – Take advantage of MLIR bufferization process
Acknowledgement

This work is supported by IO-SEA under the European High-Performance Computing Joint Undertaking (JU)
Reference

- https://www.inf.ed.ac.uk/teaching/courses/ct/18-19/slides/llvm-1-intro.pdf
- https://llvm-hpc-2020-workshop.github.io/presentations/llvmhpc2020-amini.pdf