Optimizing FDTD Solvers for Electromagnetics: A Compiler-Guided Approach with High-Level Tensor Abstractions

Yifei He, Måns I. Andersson, and Stefano Markidis

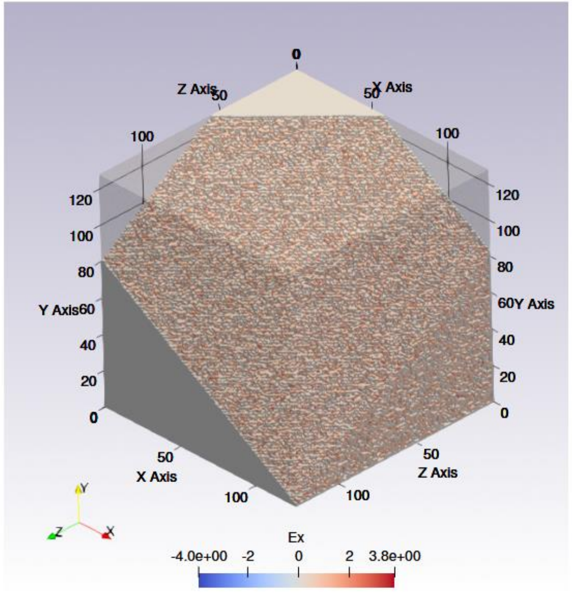
KTH Royal Institute of Technology

Outline

- Background
- Methodology
- Evaluation
- Conclusion

Background

Background : Mathematical Formulation of the FDTD Algorithm



$$\frac{\partial \mathbf{E}}{\partial t} = \frac{1}{\epsilon}\left(\nabla \times \mathbf{H} - \mathbf{J}\right), \quad \frac{\partial \mathbf{H}}{\partial t} = \frac{1}{\mu}\nabla \times \mathbf{E}$$

$$E_x^{n+1/2} \approx E_x^n + \frac{\Delta t}{\epsilon}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z}\right), \quad H_x^{n+1} \approx H_x^n - \frac{\Delta t}{\mu}\left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}\right)$$

$$E_y^{n+1/2} \approx E_y^n + \frac{\Delta t}{\epsilon}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x}\right), \quad H_y^{n+1} \approx H_y^n - \frac{\Delta t}{\mu}\left(\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x}\right)$$

$$E_z^{n+1/2} \approx E_z^n + \frac{\Delta t}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right), \quad H_z^{n+1} \approx H_z^n - \frac{\Delta t}{\mu}\left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right)$$

Methedology

Methodology: FDTD algorithm implemented in naïve Python and NumPy

```
while (t < T):  # Loop over time steps
    # Compute curl for H components
    curl_Hx(Hx,Hy,Hz,Ex,Ey,Ez)
    curl_Hy(Hx,Hy,Hz,Ex,Ey,Ez)
    curl_Hz(Hx,Hy,Hz,Ex,Ey,Ez)
    # Apply boundary conditions for H-field
    handle_H_edge(Hx,Hy,Hz,Ex,Ey,Ez)

    # Compute curl for E components
    curl_Ex(Hx,Hy,Hz,Ex,Ey,Ez)
    curl_Ey(Hx,Hy,Hz,Ex,Ey,Ez)
    curl_Ez(Hx,Hy,Hz,Ex,Ey,Ez)
    # Apply boundary conditions for E-field
    handle_E_edge(Hx,Hy,Hz,Ex,Ey,Ez)

    # Update time step
    t += dt
```

Listing 1.1: Full FDTD algorithm in Python

```
def curl_Hx(Hx, Ey, Ez): # Compute curl
  for i in range(Nx):  # Loop over x
    for j in range(Ny):  # Loop over y
      for k in range(Nz):  # Loop over z
        # Update Hx
        Hx[i, j, k] += (dt / mu0) * (
            (Ey[i, j, k+1] - Ey[i, j, k]) / Dz
          - (Ez[i, j+1, k] - Ez[i, j, k]) / Dy)
```

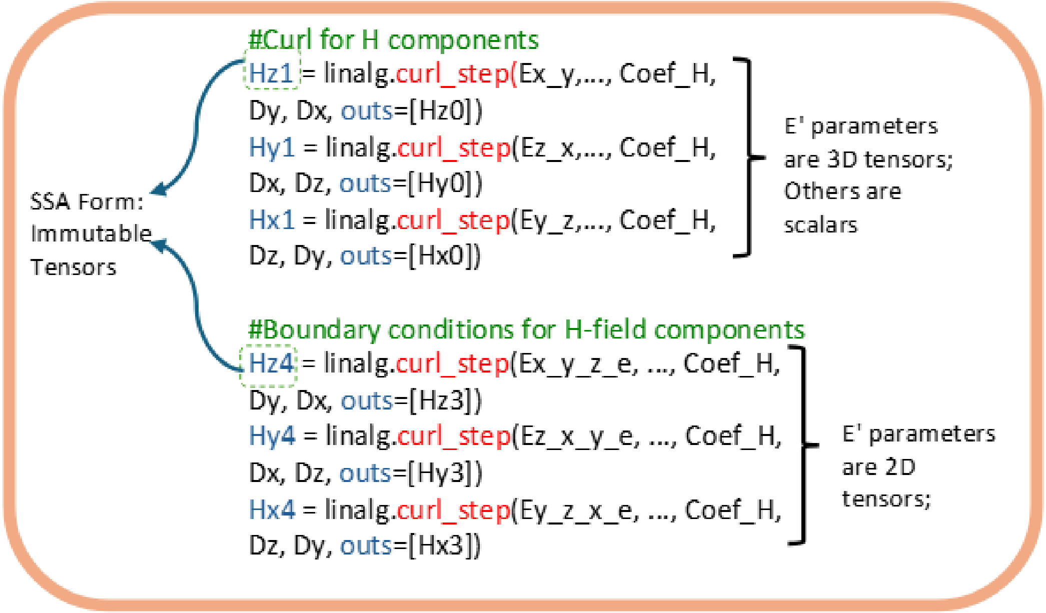
Listing 1.2: Naive Python: curl of Hx

```
def curl_slice_Hx(Hx, Ey, Ez):
    # Update Hx with NumPy slicing
    Hx[:,:,:] += (dt / mu0) * (
        (Ey[:,:,1:] - Ey[:,:,:-1]) / Dz  \
      - (Ez[:,1:,:] - Ez[:,:-1,:]) / Dy)
```

Listing 1.3: NumPy-based: curl of Hx

Methodology:

A FDTD Program in Tensor Representation (Linalg Dialect)



Methodology:

B Transform IR for Optimization Pipeline

```
# Match the curl operations
curl_op = MatchOp.match_op_names(Target,
[cul_name])
# Tiling
tiledx, loopx = TileUsingForallOp(curl_op[0], TILE_SIZE)
tiledy, loopy = TileUsingForallOp(curl_op[1], TILE_SIZE)
...
# Loop fusion
Fused_loop= loop.loop_fuse_sibling(..., target = loopx,
source = loopy)
...
# Vectorization
vf = VectorizeChildrenAndApplyPatternsOp(...,)
# Post-vec cleanup and optimization passes.
...
```

Methodology:

- The innermost dimension is set to the SIMD width

C Tiling

```
# Before tiling
%0 = linalg.curl_step ins(%ext, ..., %cst_0: ...)
outs(%ext_0 : tensor<256x256x256xf32>)

# After tiling
%0 = scf.forall (%arg6, %arg7, %arg8) in (256, 256, 16)
shared_outs(%arg9 = %ext_1) -> (tensor<256x256x256xf32>) {
# Subtensor Extraction Based on Tile Size
%ext_2 = tensor.extract_slice %ext_1[%arg6, %arg7, %12] [1, 1, 16] :
tensor<256x256x256xf32> to tensor<1x1x16xf32>
...
# Tiled Curl Operator for Smaller Sizes
%1 = linalg.curl_step ins(%ext_2, ..., %cst, ..., :
...) outs(%ext_3 : tensor<1x1x16xf32>)
#Reinserting Subtensors into the Original Tensor
scf.forall.in_parallel {
tensor.parallel_insert_slice %1 into %arg9[%arg6, %arg7, %12] [1, 1, 16]:
tensor<1x1x16xf32> into tensor<256x256x256xf32>}}
```

Methodology:

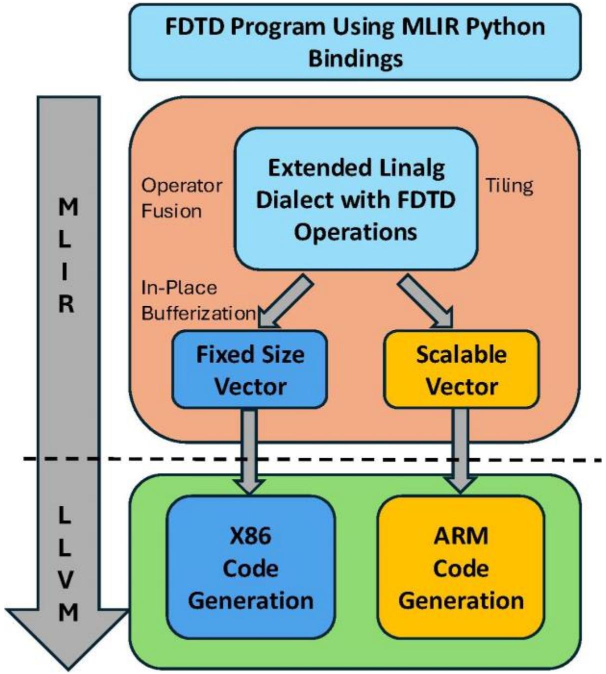
- Fixed-size vectorization for x86
- Fixed-size and scalable vectorization for ARM

D Vectorization

```
#map = affine_map<(d0) -> (d0 * 16 + 1)>
scf.forall (%arg6, %arg7, %arg8) in (256, 256, 16) {
# Memory load with affine map
%0 = affine.apply #map(%arg8)
%1 = vector.load %arg1[%arg6, %arg7, %0] :
memref<258x257x258xf32>, vector<16xf32>
...
# Arithmetic operations
%8 = arith.subf %1, %3 : vector<16xf32>
%9 = arith.divf %8, %cst_7 : vector<16xf32>
%10 = arith.subf %5, %6 : vector<16xf32>
...
# Memory store with affine map
vector.store %39, %arg4[%arg6, %arg7, %2] :
memref<256x257x256xf32>, vector<16xf32>}
```

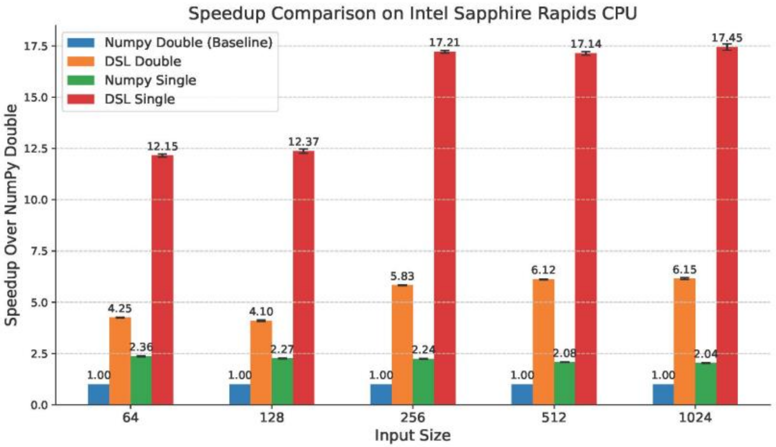
Methodology: Overview

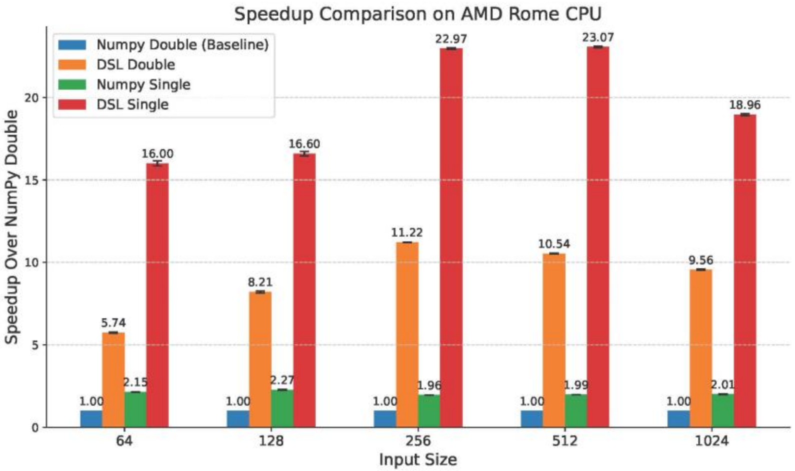
- FDTD-specific operators that encode domain semantics.
- In place bufferization
- Scalable and fixed-size vectorization
- LLVM-based code generation for multiple targets



Evaluation

Evaluation: Performance Results

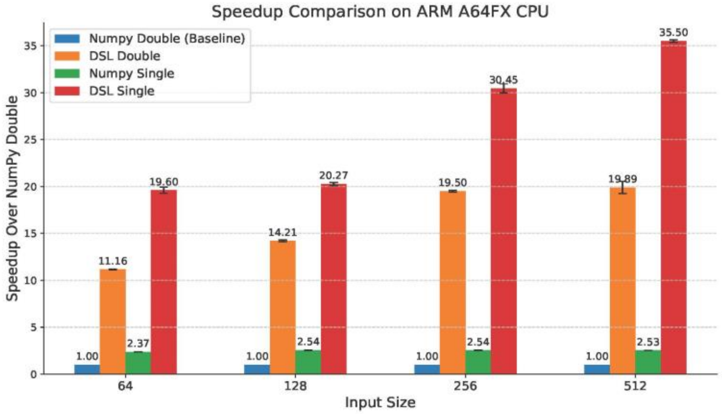




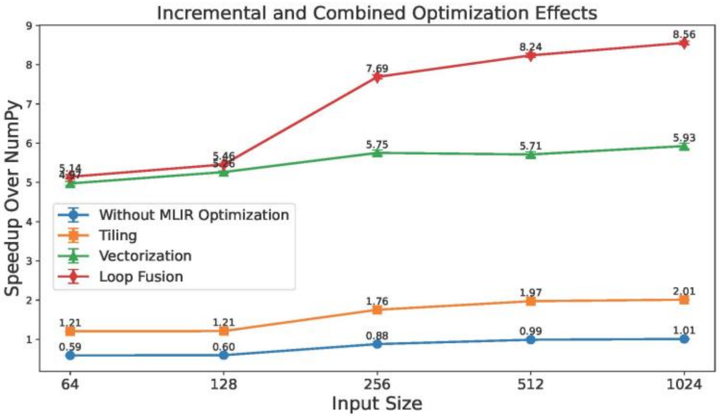
Intel Sapphire Rapids

AMD Rome

Evaluation: Performance Results



ARM A64FX



Different MLIR Optimization Combinations

Evaluation: Performance Results

Profiling Results of Optimization Combinations
for N = 256 on Intel CPU (Single Precision)

Workloads	Vectorization Ratio & Type	L1 Cache Loads	L1 Cache Load Misses	LLC Loads	LLC Load Misses	Speedup
Numpy	99.7% AVX256	1x	9.04%	1x	47.21%	1x
MLIR: Fusion&Vec&Tiling	98.0% AVX512	0.11x	4.96%	0.03x	62.94%	7.69x
MLIR: Vec&Tiling	98.0% AVX512	0.10x	29.10%	0.21x	54.57%	5.75x
MLIR: Tiling	0% Scalar	1.69x	0.13%	0.02x	68.85%	1.76x
MLIR: No-Opt	0% Scalar	4.59x	0.04%	0.02x	69.12%	0.88x

Conclusion

Conclusion

- High-level tensor abstractions for FDTD kernels enable automatic optimizations such as tiling and fusion by leveraging tensor expressions combined with domain-specific knowledge of FDTD.

- Automated extraction of hardware-specific parallelism, integrating vectorization and architecture-aware code generation for Intel, AMD, and ARM CPUs through a unified MLIR/LLVM backend.

- Performance evaluation and analysis of our end-to-end domain-specific compiler for the FDTD solver on Intel, AMD, and ARM CPUs, achieving up to 10 speedup over the baseline NumPy implementation.

Q&A