Introduction to Modern Compilers

Dept of CSA
Indian Institute of Science

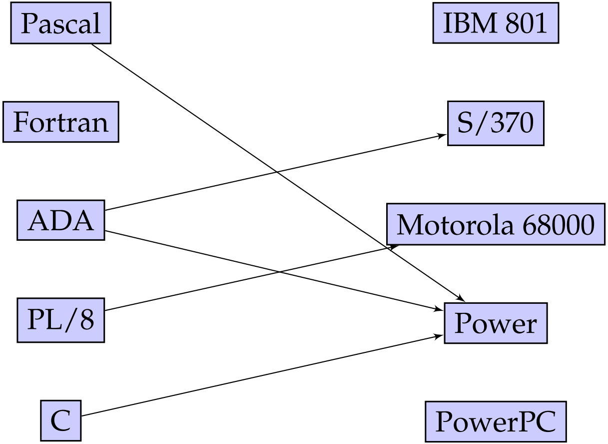
udayb@iisc.ac.in

https://www.csa.iisc.ac.in/~udayb

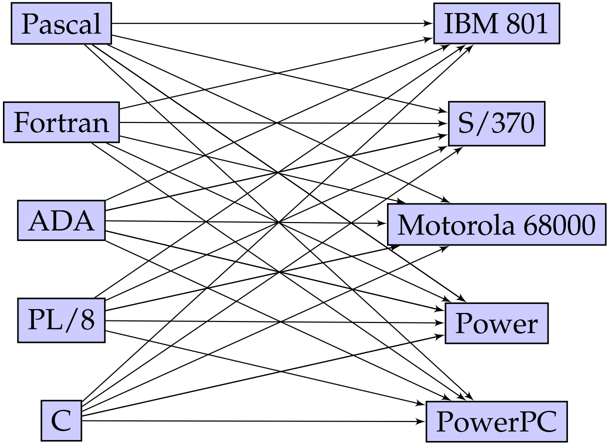
Compilers for AI

- Compilers are language translators: they translate programming languages to instructions hardware can execute
- Everything is compiled (directly or indirectly): operating systems, databases, compilers, ...
- Compilers can be for programming languages, programming models or frameworks embedded in existing languages
- Why and when do we need new compilers?

Compilers - The Early Days

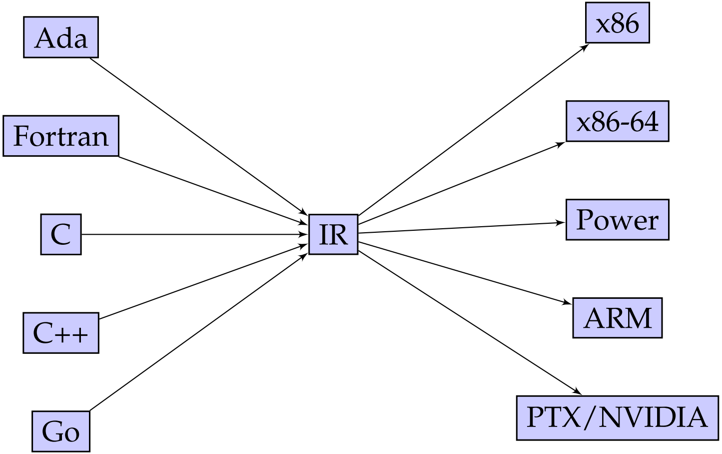


COMPILERS - THE EARLY DAYS



- M **languages,** N **targets** $\Rightarrow M * N$ **compilers! Not scalable!**

Compilers Evolution - $M + N$



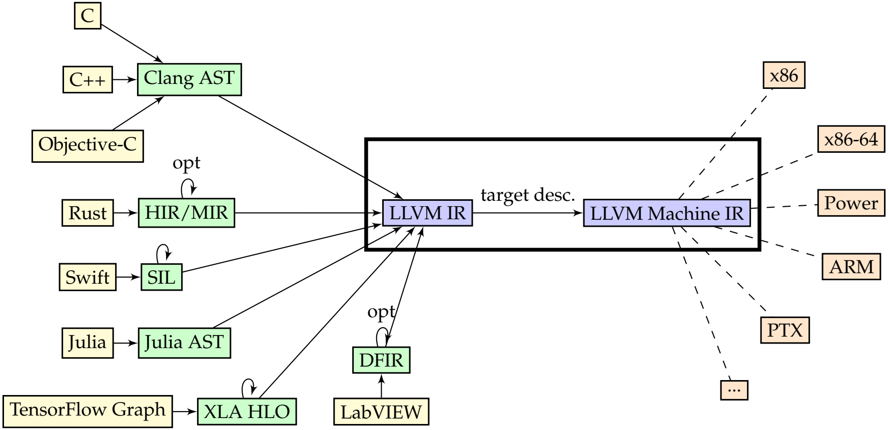
- **With an common IR, we have $M + N + 1$ compilers!**

What does an IR look like?

- A representation convenient to analyze and transform
- Round-trippable form that you can parse and print
- Low-level IRs are three-address code-like
- IRs have used expressions trees, 3-address code, graphs.
- Static Single Assignment: a property of IRs that makes it convenient; most IRs now use SSA

```
define void @foo(ptr nocapture %a) {
entry:
  br label %for.body

for.body:   ; preds = %for.body, %entry
  %indvars.iv = phi i64 [ 0, %entry ], [ %indvars.iv.next, %for.body ]
  %arrayidx = getelementptr inbounds i32, ptr %a, i64 %indvars.iv
  %0 = load i32, ptr %arrayidx, align 4
  %1 = add i32 %0, 2
  %inc = add nsw i32 %0, 1
  store i32 %inc, ptr %arrayidx, align 4
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond = icmp eq i64 %indvars.iv.next, 64
  br i1 %exitcond, label %for.end, label %for.body

for.end:          ; preds = %for.body
  ret void
}
```

Modern Compilers - LLVM IR-based



- **LLVM: modular, reusable, open-source — even better:** $M + n + 1$

Compilers for AI

ML/AI programming frameworks

. . . ?

Compiler infrastructure?

Explosion of AI chips and accelerators

- Space in between is ruled by hand-written libraries. Not scalable.
- The right compiler tools weren't available until 2019.

A Python-based AI framework

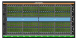
- Where does performance in Python-based frameworks come from?
- Largely from libraries written in C, C++, CUDA, and even assembly
- Compilers exist: **XLA, TorchInductor (torch.compile)**, TensorRT
  - Limited in many ways
  - Still evolving

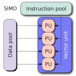
```
class SelfAttentionLayer(nn.Module):
    def __init__(self, feature_size):
        super(SelfAttentionLayer, self).__init__()
        self.feature_size = feature_size

        # Linear transformations for Q, K, V from the same source.
        self.key = nn.Linear(feature_size, feature_size)
        self.query = nn.Linear(feature_size, feature_size)
        self.value = nn.Linear(feature_size, feature_size)

    def forward(self, x, mask=None):
        # Apply linear transformations.
        keys = self.key(x)
        queries = self.query(x)
        values = self.value(x)

        # Scaled dot-product attention.
        scores = torch.matmul(queries, keys.transpose(-2, -1))
            / torch.sqrt(torch.tensor(self.feature_size, dtype=torch.float32))

        # Apply mask (if provided).
        if mask is not None:
            scores = scores.masked_fill(mask == 0, -1e9)

        # Apply softmax.
        attention_weights = F.softmax(scores, dim=-1)

        # Multiply weights with values.
        output = torch.matmul(attention_weights, values)

        return output, attention_weights
```

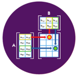
How is hardware evolving?

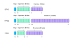
- Multiple cores
- Wider SIMD
- Many cores
- Heterogeneity
- Tensor/matmul cores

From 2000s to now





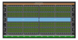


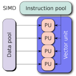


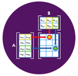
How is hardware evolving?

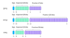
- Multiple cores
- Wider SIMD
- Many cores
- Heterogeneity
- Tensor/matmul cores
- Low-precision compute instructions

From 2000s to now









HOW ARE PROGRAMMING FRAMEWORKS EVOLVING?

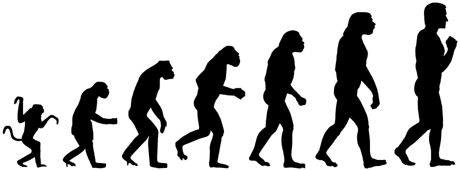
ML/AI programming frameworks

- Programmer productivity
- Write less and do more
- Hardware usability
- Deliver performance
- Deliver portability

Big Picture: Role of Compilers

General-purpose: Evolutionary

- Improve existing **general-purpose** compilers (for C, C++, Rust, ...)
- Programmers have a lot of control and complexity
- Limited improvements but wide impact



Domain-specific: Revolutionary

- Build new **domain-specific languages and compilers**
- Programmers say **WHAT** and not **HOW** they execute
- Dramatic speedups

- **Important to pursue both**

Compilers for AI

ML/AI programming frameworks

. . . **?**

Compiler infrastructure?

Explosion of AI chips and accelerators

- I was visiting Google in 2018 — to tackle TensorFlow compilation for TPUs

ML/AI programming frameworks

. . . **?**

Compiler infrastructure?

Explosion of AI chips and accelerators

- Realization that a brand new IR was needed

Compilers for AI

ML/AI programming frameworks

. . . ?

Explosion of AI chips and accelerators

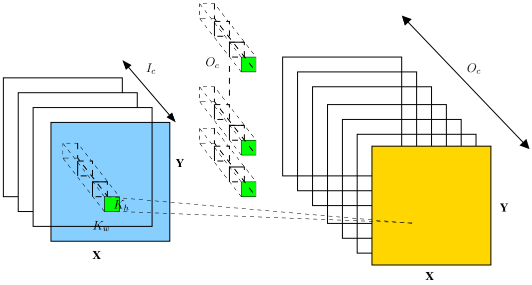
- MLIR infrastructure: open-sourced by Google in 2019

MLIR

- Requirements
  - Loops and multi-dimensional arrays (tensors) had to be first class citizens
  - Had to be extensible (types, operations, attributes)
  - Had to enable building both general-purpose and domain-specific compilers and even more.
  - Had to be open-source with a permissive license
- ML in MLIR: Multi-level

Multi-dimensionality everywhere: CNN Convolution

```
for (n = 0; n < N; n++) // Samples in a batch.
 for (o = 0; o < Oc; o++) // Output feature channels.
  for (i = 0; i2 < Ic; i++) // Input feature channels.
   for (y = 0; i3 < Y; i3++) // Layer height.
    for (x = 0; i4 < X; i4++) // Layer width.
     for (kh = 0; i5 < Kh; i5++) // Convolution kernel height.
      for (kw = 0; i6 < Kw; i6++) // Convolution kernel width.
       output[n, o, y, x] += input[n, i, y+kh, x+kw] * weights[o, i, kh, kw];
```

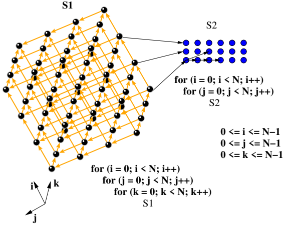


MLIR: Multi-level Intermediate Representation

1. Ops (general purpose to domain specific) on tensor types / graph form

```
%patches = "tf.reshape"(%patches, %minus_one, %minor_dim_size)
                   : (tensor<? x ? x ? x ? x f32>, index, index) −> tensor<? x ? x f32>
%mat_out = "tf.matmul"(%patches_flat, %patches_flat){transpose_a : true}
                   : (tensor<? x ? x f32>, tensor<? x ? x f32>) −> tensor<? x ? x f32>
%vec_out = "tf.reduce_sum"(%patches_flat) {axis: 0} : (tensor<? x ? x f32>) −> tensor<? x f32>
```

2. Loop-level / mid-level form




```
affine.for %i = 0 to 8 step 4 {
  affine.for %j = 0 to 8 step 4 {
    affine.for %k = 0 to 8 step 4 {
      affine.for %ii = #map0(%i) to #map1(%i) {
        affine.for %jj = #map0(%j) to #map1(%j) {
          affine.for %kk = #map0(%k) to #map1(%k) {
            %5 = affine.load %arg0[%ii, %kk] : memref<8 x 8 x vector<64 x f32>>
            %6 = affine.load %arg1[%kk, %jj] : memref<8 x 8 x vector<64 x f32>>
            %7 = affine.load %arg2[%ii, %jj]  : memref<8 x 8 x vector<64 x f32>>
            %8 = arith.mulf %5, %6 : vector<64xf32>
            %9 = arith.addf %7, %8 : vector<64xf32>
            affine.store %9, %arg2[%ii, %jj]  : memref<8 x 8 x vector<64xf32>>
          }
        }
      }
    }
  }
}
```

3. Low-level form: closer to hardware

```
%v1 = memref.load %a[%i2, %i3] : memref<256 x 64 x vector<16 x f32>>
%v2 = memref.load %b[%i2, %i3] : memref<256 x 64 x vector<16 x f32>>
%v3 = addf %v1, %v2 : vector<16 x f32>
memref.store %v3, %d[%i2, %i3] : memref<256 x 64 x vector<16 x f32>>
```

Polyhedral Framework

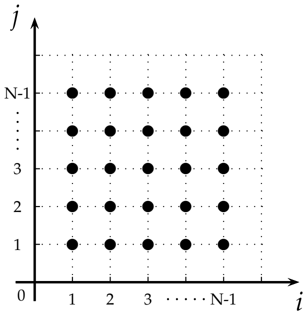
```
for (t = 0; t < T; t++)
 for (i = 1; i < N+1; i++)
  for (j = 1; j < N+1; j++)
   A[(t+1)%2][i][j] = f((A[t%2][i+1][j], A[t%2][i][j], A[t%2][i-1][j],
     A[t%2][i][j+1], A[t%2][i][j-1]);
```

1. **Domains**
   - Every statement has a domain or an index **set –** instances that have to be executed
   - Each instance is a vector (of loop index values from outermost to innermost) $D_S = \{[t,i,j] \mid 0 \leq t \leq T-1,\ 1 \leq i,j \leq N\}$
2. **Dependences**
   - A dependence is a **relation** between domain / index set instances that are in conflict (more on next slide)
3. **Schedules**
   - are **functions** specifying the *order* in which the domain instances should be executed
   - Specified statement-wise and **typically** one-to-one
   - $T((i,j)) = (i+j,j)$ or $\{[i,j] \rightarrow [i+j,j] \mid \ldots\}$

Domains, Dependences, and Schedules

```
for (i=1; i<=N-1; i++)
 for (j=1; j<=N-1; j++)
  A[i][j] = A[i-1][j] + A[i][j-1]);
```

```
for (t1=2;t1<=2*N-2;t1++) {
#pragma omp parallel for
 for (t2=max(1,t1-N+1);t2<=min(N-1,t1-1);t2++) {
  a[(t1-t2)][t2] = a[(t1-t2) - 1][t2] + a[(t1-t2)][t2 - 1];
 }
}
```



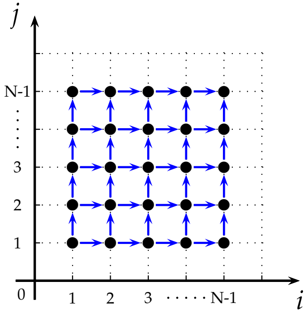
Original space (i, j)

- **Domain**: $\{[i, j] \mid 1 \leq i, j \leq N - 1\}$

Domains, Dependences, and Schedules

```
for (i=1; i<=N-1; i++)
  for (j=1; j<=N-1; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]);
```

```
for (t1=2;t1<=2*N-2;t1++) {
#pragma omp parallel for
  for (t2=max(1,t1-N+1);t2<=min(N-1,t1-1);t2++) {
    a[(t1-t2)][t2] = a[(t1-t2) - 1][t2] + a[(t1-t2)][t2 - 1];
  }
}
```



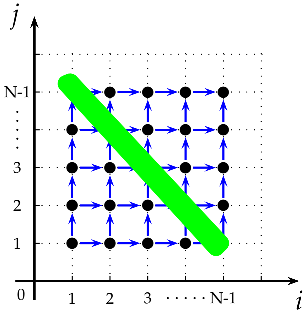
Original space (i, j)

- **Dependences**:
  1. $\{[i,j] \to [i+1,j] \mid 1 \le i \le N-2, 0 \le j \le N-1\}$ — **(1,0)**
  2. $\{[i,j] \to [i,j+1] \mid 1 \le i \le N-1, 0 \le j \le N-2\}$ — **(0,1)**

Domains, Dependences, and Schedules

```
for (i=1; i<=N-1; i++)
  for (j=1; j<=N-1; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]);
```

```
for (t1=2;t1<=2*N-2;t1++) {
#pragma omp parallel for
  for (t2=max(1,t1-N+1);t2<=min(N-1,t1-1);t2++) {
    a[(t1-t2)][t2] = a[(t1-t2) - 1][t2] + a[(t1-t2)][t2 - 1];
  }
}
```



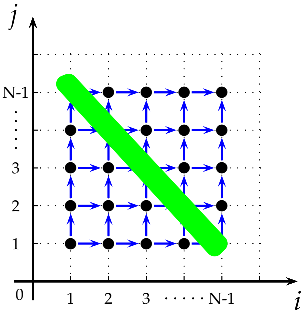
Original space (i, j)

- **Dependences**:
  1. $\{[i,j] \rightarrow [i+1,j] \mid 1 \leq i \leq N-2, 0 \leq j \leq N-1\}$ — **(1,0)**
  2. $\{[i,j] \rightarrow [i,j+1] \mid 1 \leq i \leq N-1, 0 \leq j \leq N-2\}$ — **(0,1)**

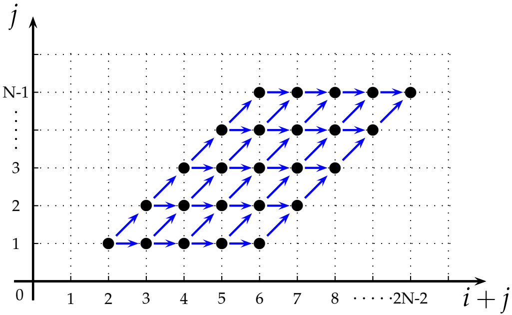
Domains, Dependences, and Schedules

```
for (i=1; i<=N-1; i++)
  for (j=1; j<=N-1; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]);
```

```
for (t1=2;t1<=2*N-2;t1++) {
#pragma omp parallel for
  for (t2=max(1,t1-N+1);t2<=min(N-1,t1-1);t2++) {
    a[(t1-t2)][t2] = a[(t1-t2) - 1][t2] + a[(t1-t2)][t2 - 1];
  }
}
```



Original space (i, j)



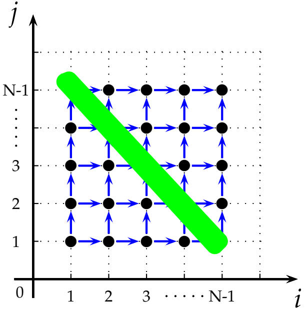
Transformed space $(i + j, j)$

- **Schedule**: $T(i, j) = (i + j, j)$
  - Dependences: (1,0) and (0,1) now become (1,0) and (1,1) resp.
  - Inner loop is now parallel

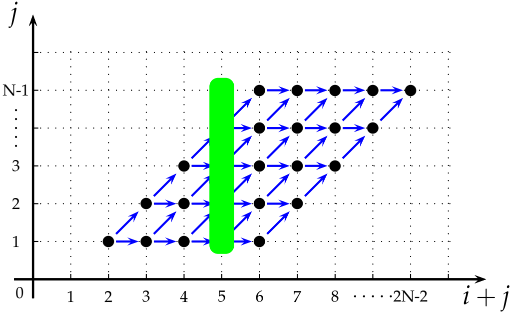
DOMAINS, DEPENDENCES, AND SCHEDULES

```
for (i=1; i<=N-1; i++)
  for (j=1; j<=N-1; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]);
```

```
for (t1=2;t1<=2*N-2;t1++) {
#pragma omp parallel for
  for (t2=max(1,t1-N+1);t2<=min(N-1,t1-1);t2++) {
    a[(t1-t2)][t2] = a[(t1-t2) - 1][t2] + a[(t1-t2)][t2 - 1];
  }
}
```



Original space (i, j)



Transformed space $(i + j, j)$

- **Schedule**: $T(i,j) = (i+j, j)$
  - Dependences: (1,0) and (0,1) now become (1,0) and (1,1) resp.
  - Inner loop is now parallel

Modern Compiler Topics in this course

1. Compiler optimizations for parallelism and locality
2. Affine abstraction/Polyhedral framework (only the basics)
3. MLIR
4. Practice: Building compilers using MLIR
5. Practice: Building compilers and optimizers for AI frameworks (basic overview, pointers)
6. Foundations: SSA, Dominance, Basic concepts for control flow analysis, data flow analysis, ...