Automatic Generation of Multi-Objective Polyhedral Compiler Transformations

Lorenzo Chelini, Tobias Gysi, Tobias Grosser, Martin Kong and Henk Corporaal

5th October 2020

Project 676240
MSCA-ITN-EID

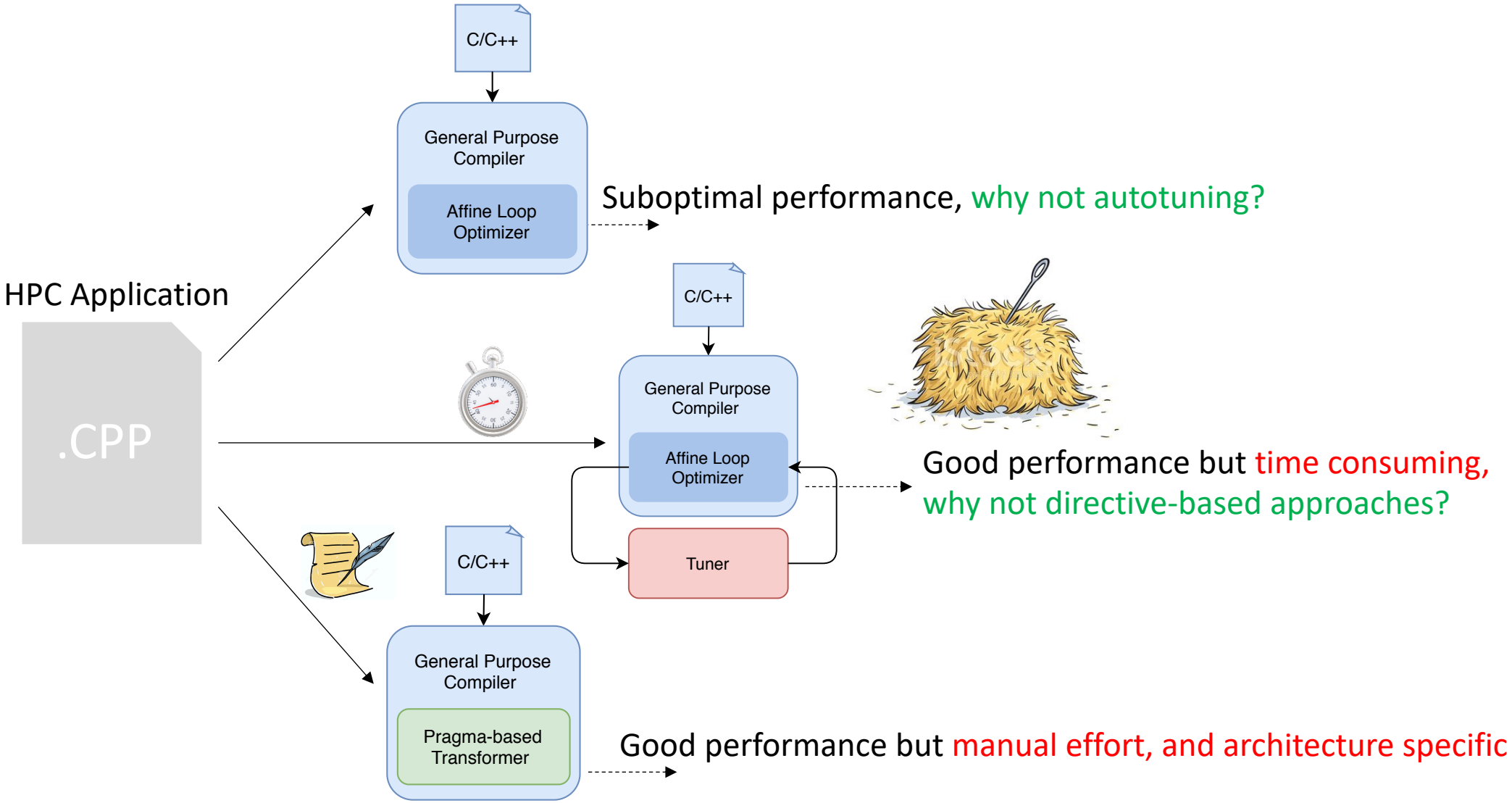
Talk Outline

- Challenges and Motivation
- Adaptive Scheduler
- Results
- Conclusion

Talk Outline

- **Challenges and Motivation**
- Adaptive Scheduler
- Results
- Conclusion

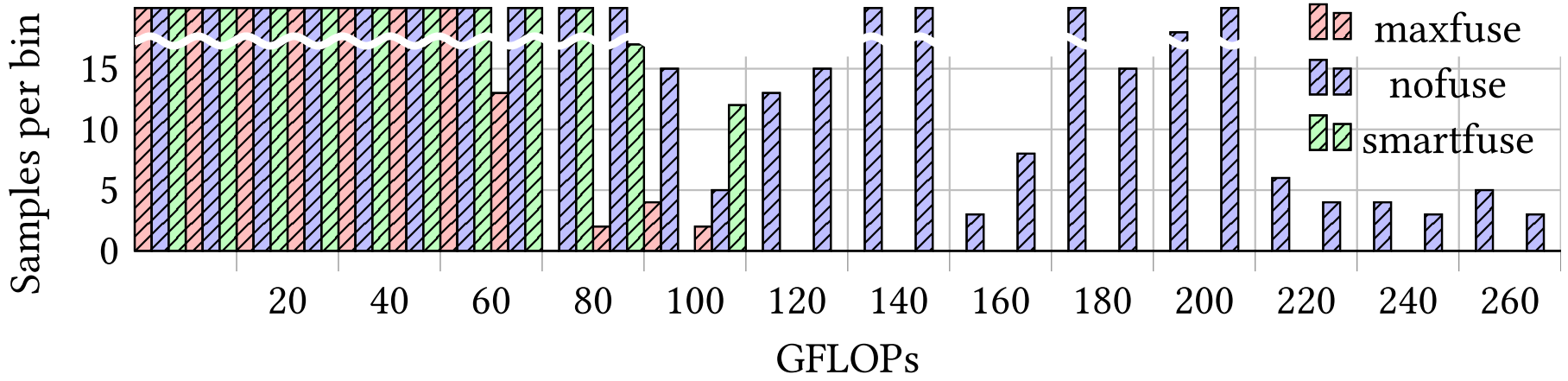
Challenges


HPC Application
.CPP
C/C++
General Purpose Compiler
Affine Loop Optimizer
Suboptimal performance, why not autotuning?
C/C++
General Purpose Compiler
Affine Loop Optimizer
Tuner
Good performance but time consuming, why not directive-based approaches?
C/C++
General Purpose Compiler
Pragma-based Transformer
Good performance but manual effort, and architecture specific

Motivating Example

- Baseline: original and variants generated with the Pluto compiler
- More than 2000 variants
- Only 3/2000 variants reach 260 GFLOP/s
- MKL 0.75 TFLOP/s

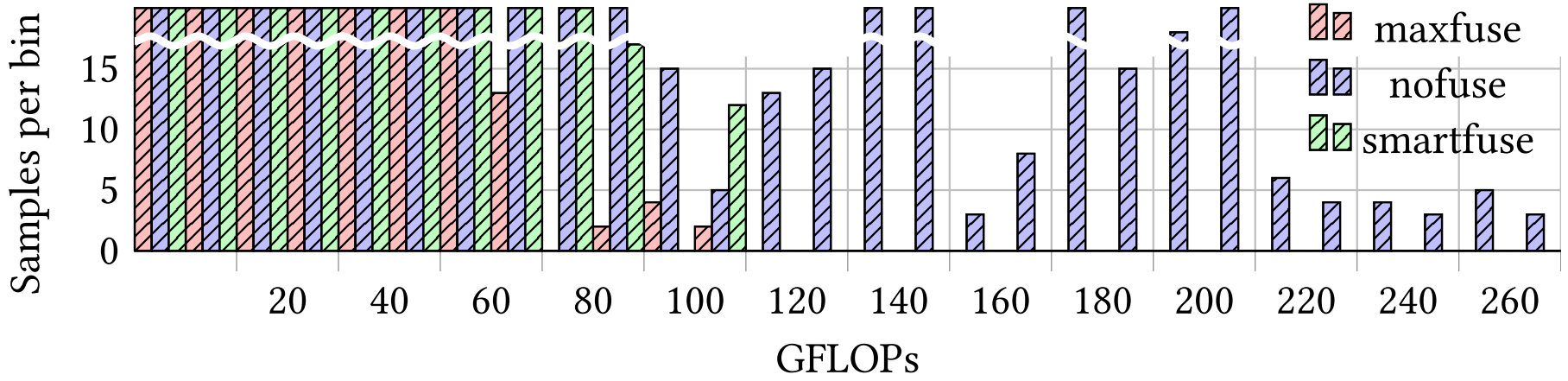
```
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S1:     tmp[i][j] = 0;
        for (int k = 0; k < N; ++k)
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
    }
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S3:     D[i][j] *= beta;
        for (int k = 0; k < N; ++k)
S4:       D[i][j] += tmp[i][k] * C[k][j];
    }
```



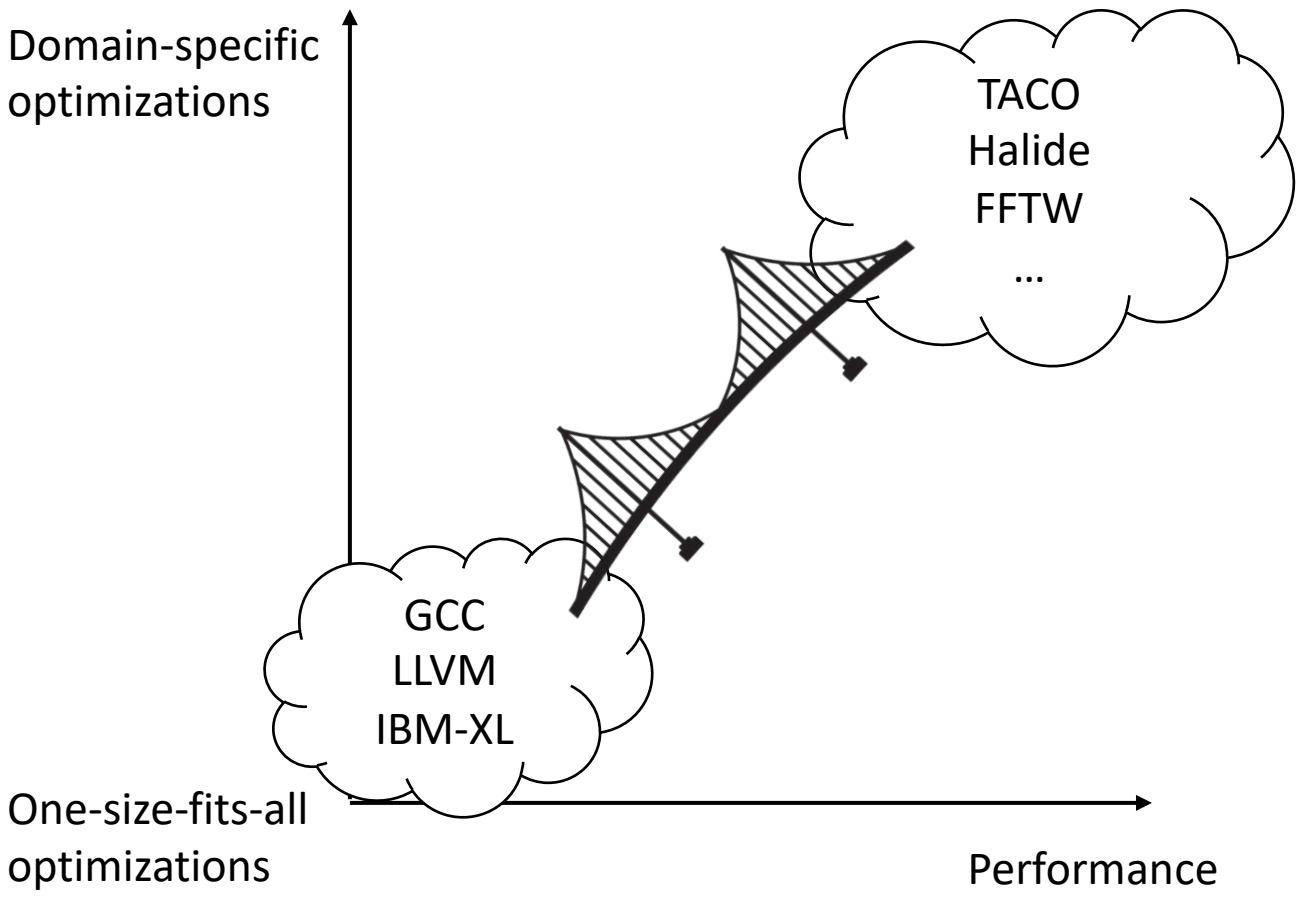
Motivating Example

- Baseline: original and variants generated with the Pluto compiler
- More than 2000 variants
- Only 3/2000 variants reach 260 GFLOP/s
- MKL 0.75 TFLOP/s

- Our work:
  - Achieves almost the same performance (214 GFLOP/s)
  - By evaluating **only** five variants



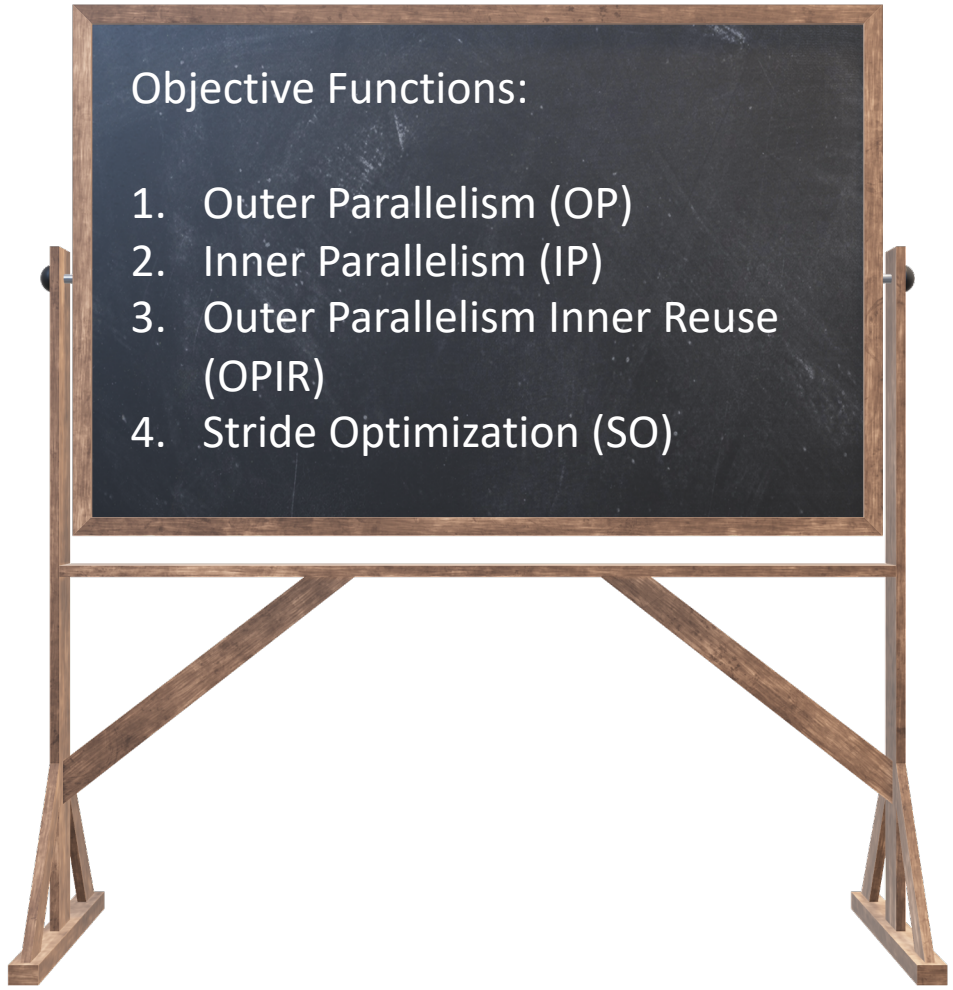
Fundamental Contribution



This Work

- ✓ Bridge the gap between domain-specific and general-purpose optimizers
- ✓ Proposes a tractable approach to navigate large space of affine transformations
- ✓ Application and target customization

How to achieve this? Objective Functions



Stride Optimization:

```
for (int i = 0; i < NI; i++)
    for (int j = 0; j < NJ; j++)
        for (int k = 0; k < NK; k++) {

            C[i][j] = 0;     // reuse

            A[i][k] = 0;     // vectorization

            B[k][j] = 0;     // penalty

        }
```


Source: Martin Kong, and Louis-Noël Pouchet. "Model-driven transformations for multi-and many-core CPUs." *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2019.
Source: https://pliss2019.github.io/albert_cohen_slides.pdf

Objective Functions an Example

```
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S1:     tmp[i][j] = 0;
        for (int k = 0; k < N; ++k)
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
    }
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S3:     D[i][j] *= beta;
        for (int k = 0; k < N; ++k)
S4:       D[i][j] += tmp[i][k] * C[k][j];
    }
```

```
// S1 and S2 are distributed as
// a by-product of the selected objectives.

// S1 /* ... */
```

<OPIR, IP, SO>

```
for (int k = 0; k < N; ++k)      // reuse
  for (int i = 0; i < N; ++i)    // parallel
    for (int j = 0; j < N; ++j) // vectorizable
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
```

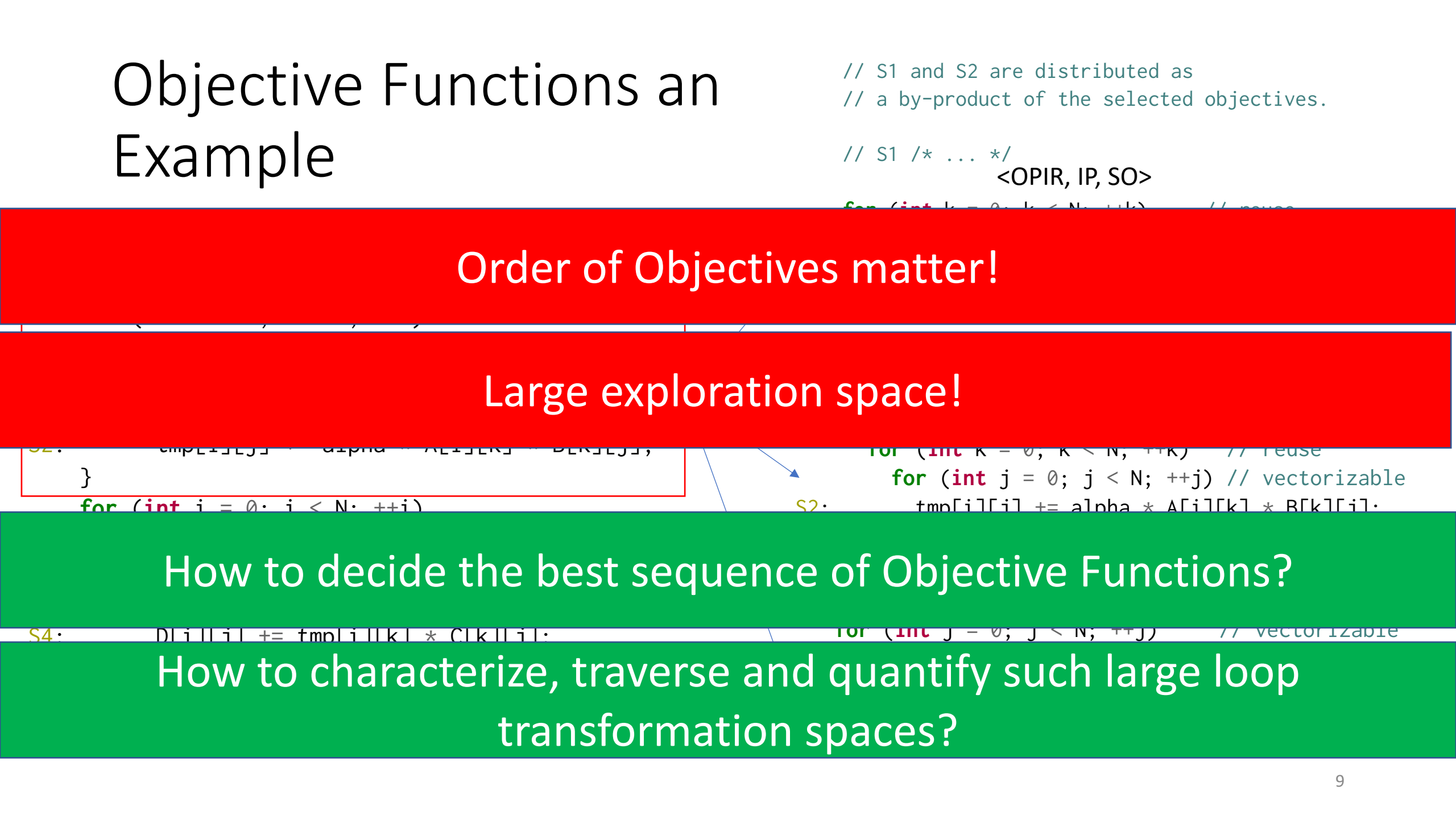
<OP, IP, SO>

```
for (int i = 0; i < N; ++i)      // parallel
  for (int k = 0; k < N; ++k)    // reuse
    for (int j = 0; j < N; ++j) // vectorizable
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
```

<IP, OPIR,OP>

```
for (int j = 0; j < N; ++j)      // vectorizable
  for (int k = 0; k < N; ++k)    // reuse
    for (int i = 0; i < N; ++i) // parallel
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
```

Objective Functions an Example

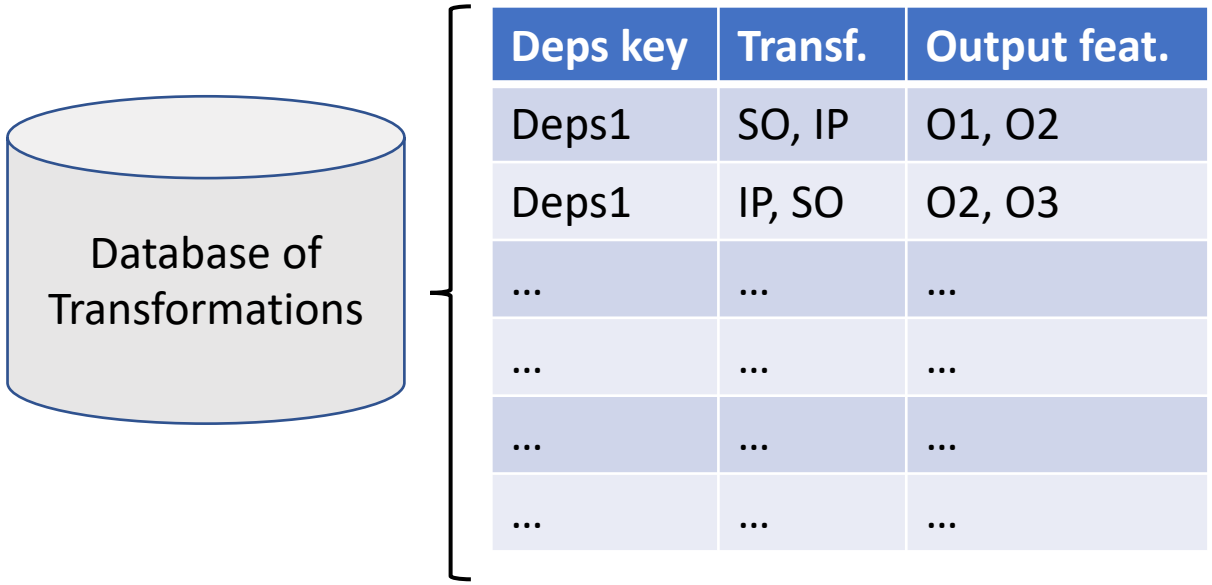


Talk Outline

- Challenges and Motivation
- **Adaptive Scheduler**
- Results
- Conclusion

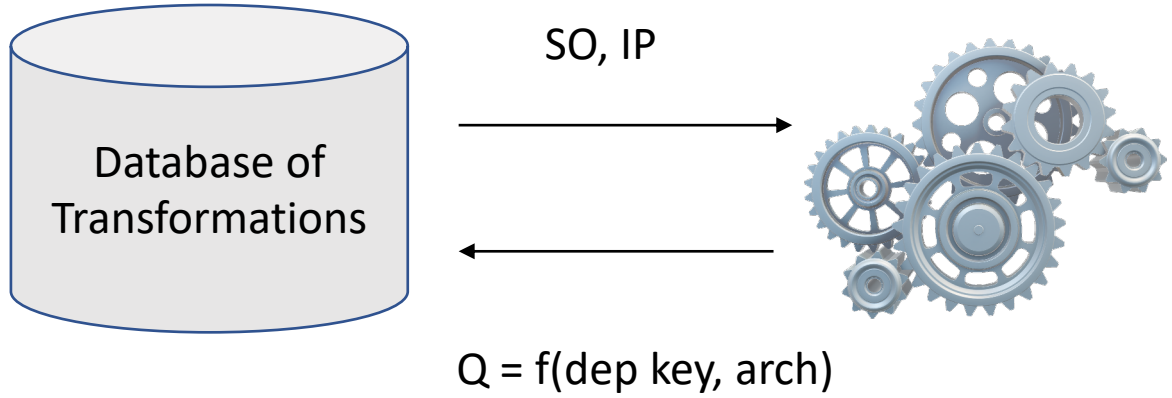
Adaptive Scheduler: Two components

- Offline Stage: Building a Database of Transformations

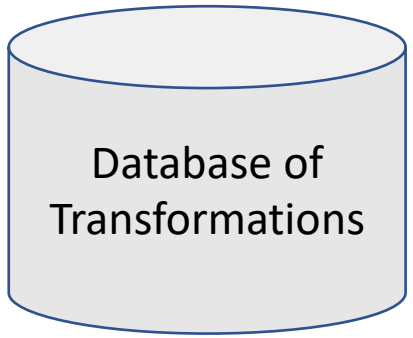


Deps key	Transf.	Output feat.
Deps1	SO, IP	O1, O2
Deps1	IP, SO	O2, O3
…	…	…
…	…	…
…	…	…
…	…	…

- Online Stage: Select transformations



Database of Transformations



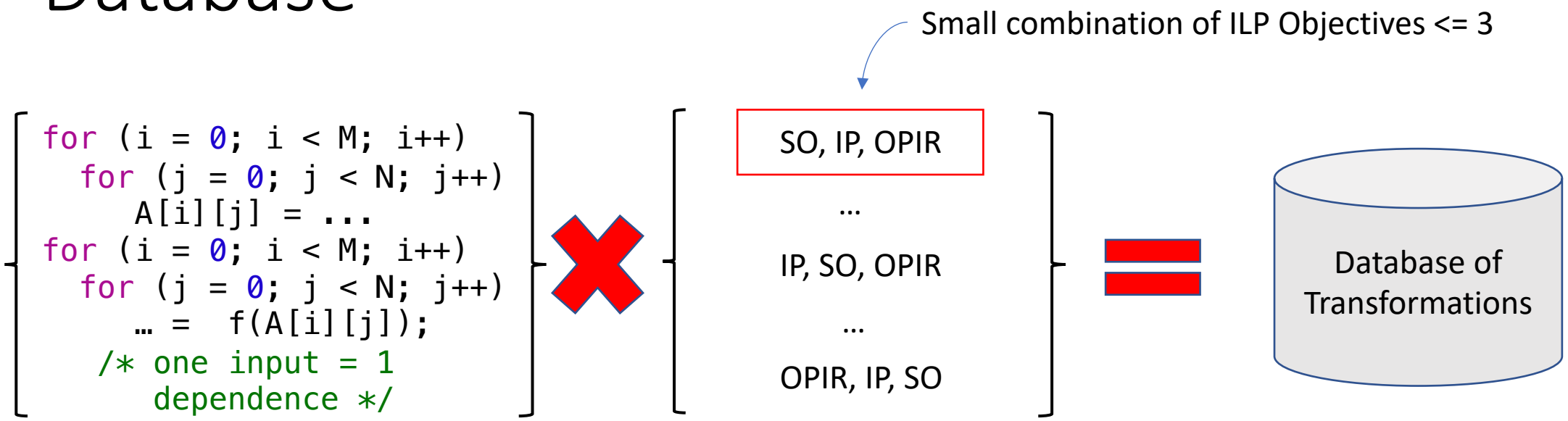
Dependences key	Transformations	Output features
Dependence 1	Stride Opt., Inner Paral.	O1, O2
Dependence 1	Inner Paral., Stride Opt.	O2, O3

- Model a single dependence polyhedron
- Extremely simple to allow scalability
- Dependences keys encompass structural information and dependence features (e.g. 1-to-1 or 1-to-N)

- Each transformation implemented as an Integer Linear Program (ILP) cost function
- Characterize combined behavior of transformations
- Order matters

- Capture transformation results and properties (e.g. number of parallel loops, total number of loops, stride access, and more)
- Used to compute a score for each candidate transformation

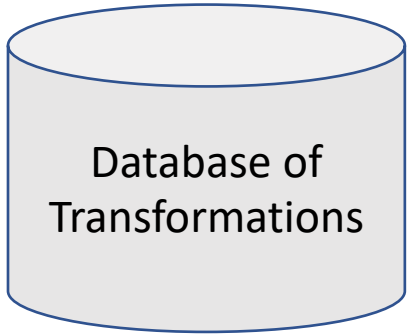
Offline Stage: Building the Transformation Database



Synthesized or extracted from larger representative applications

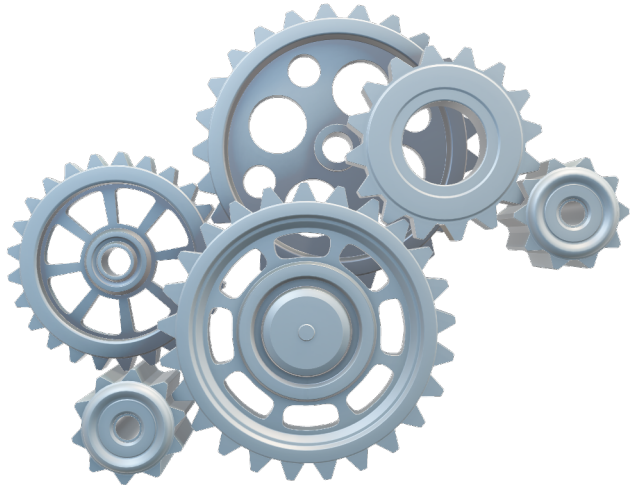
Candidate transformation is a tuple of ILP objectives, e.g. all 3 permutations in a set of 10 transformations = 720 transformation candidates

Online Stage: Selecting Transformations



Query =
f (dependence, architecture) [*]

Result = <T4, T1, T7>



Transformation = <T2, T5, T4> + <T4, T1, T7>

Query 1 Query 2

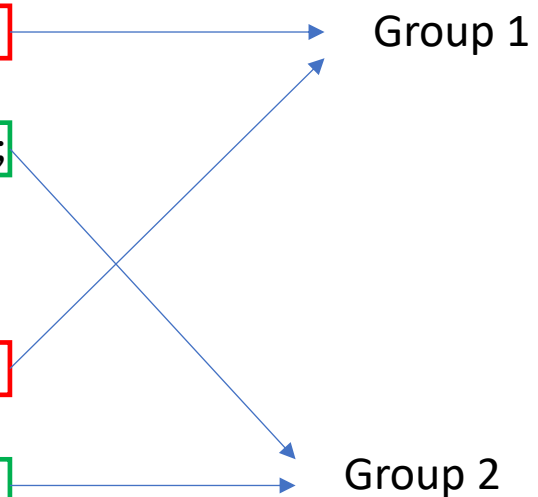
1. Divide statements into equivalence classes with 5 partition policies
2. Select most influential dependences in each partition

[*] Section 4.7

A step-by-step Example

- Step 1: Partition statements

```
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S1:     tmp[i][j] = 0;
        for (int k = 0; k < N; ++k)
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
    }
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S3:     D[i][j] *= beta;
        for (int k = 0; k < N; ++k)
S4:       D[i][j] += tmp[i][k] * C[k][j];
    }
```



A step-by-step Example

```
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S1:     tmp[i][j] = 0;
        for (int k = 0; k < N; ++k)
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
    }
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S3:     D[i][j] *= beta;
        for (int k = 0; k < N; ++k)
S4:       D[i][j] += tmp[i][k] * C[k][j];
    }
```

- Step 1: Partition statements
- Step 2: Ranking dependences

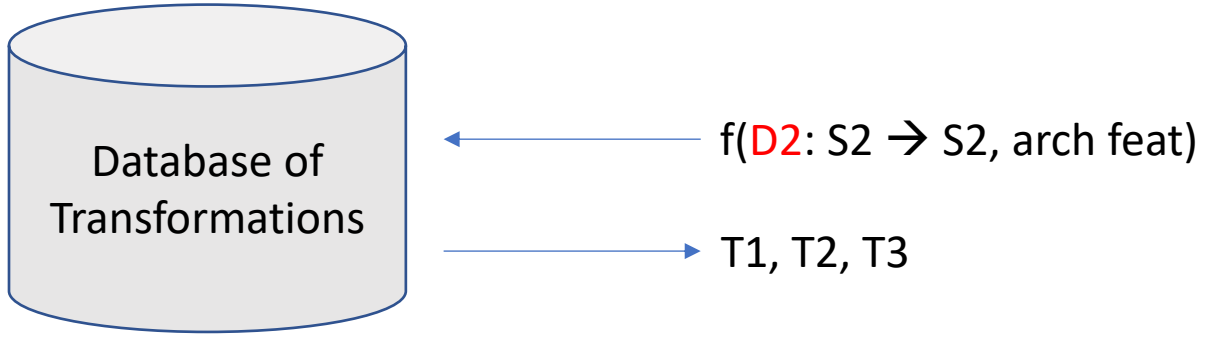
D1: S1 → S2
D2: S2 → S2
D3: S2 → S4
D4: S3 → S4
D5: S4 → S4

D2: S2 → S2
D5: S4 → S4
D3: S2 → S4
D4: S3 → S4
D1: S1 → S2

A step-by-step Example

```
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S1:     tmp[i][j] = 0;
        for (int k = 0; k < N; ++k)
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
    }
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S3:     D[i][j] *= beta;
        for (int k = 0; k < N; ++k)
S4:       D[i][j] += tmp[i][k] * C[k][j];
    }
```

- Step 1: Partition statements
- Step 2: Ranking dependences
- Step 3: Query Database



D5 already covered by D2

A step-by-step Example

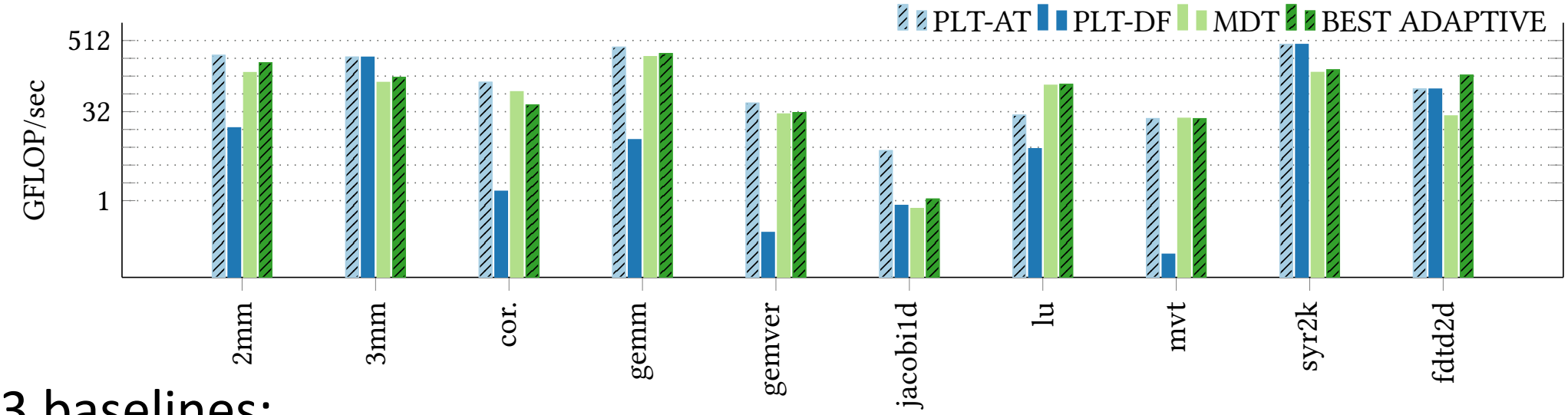
```
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S1:     tmp[i][j] = 0;
        for (int k = 0; k < N; ++k)
S2:       tmp[i][j] += alpha * A[i][k] * B[k][j];
    }
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j) {
S3:     D[i][j] *= beta;
        for (int k = 0; k < N; ++k)
S4:       D[i][j] += tmp[i][k] * C[k][j];
    }
```

- Step 1: Partition statements
- Step 2: Ranking dependences
- Step 3: Query Database
- Step 4: Compute lexicographic optimum and apply transformations

Talk Outline

- Challenges and Motivation
- Adaptive Scheduler
- **Results**
- Conclusion

Results… Some of them



- 3 baselines:
  - Pluto autotune (PLT-AT): Best of thousand program variants
  - Pluto Default (PLT-DF): Pluto out-of-the-box
  - Model driven transformations (MDT): Single optimized variant using predefined order of transformations
- Best adaptive: Best variants for the 5 proposed transformation policies
- Machine: 48-core Intel Xeon Platinum (MKL 0.75 TFLOP/s)

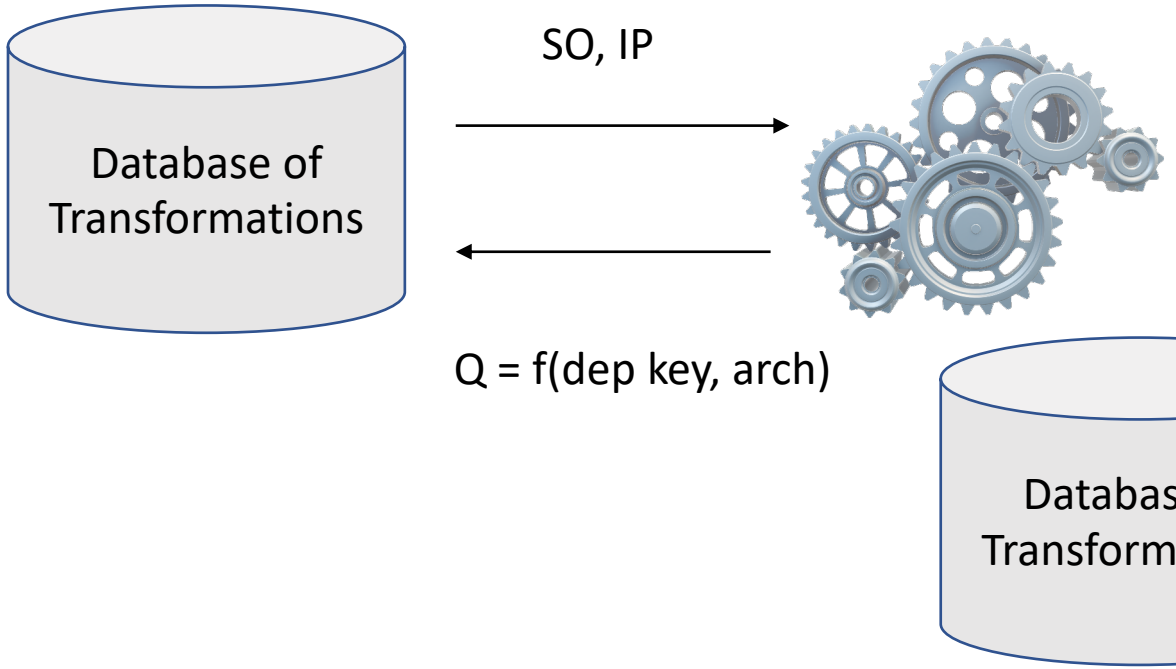
Talk Outline

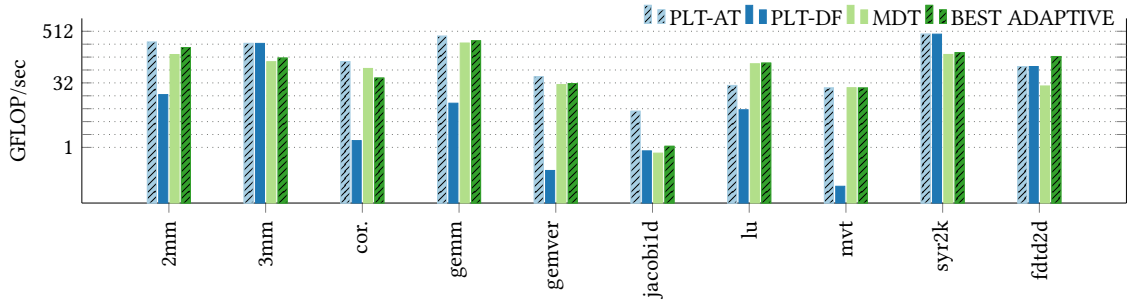
- Challenges and Motivation
- Adaptive Scheduler
- Results
- **Conclusion**

Conclusion

- Scalable approach to traverse large transformation space
- Order of objectives is not hardcoded
- Reduction of tuning time
- Competitive performance with minimal exploration

Our adaptive scheduler bridges the gap between general-purpose and domain-specific compilation flow





Deps key	Transf.	Output feat.
Deps1	SO, IP	O1, O2
Deps1	IP, SO	O2, O3
…	…	…