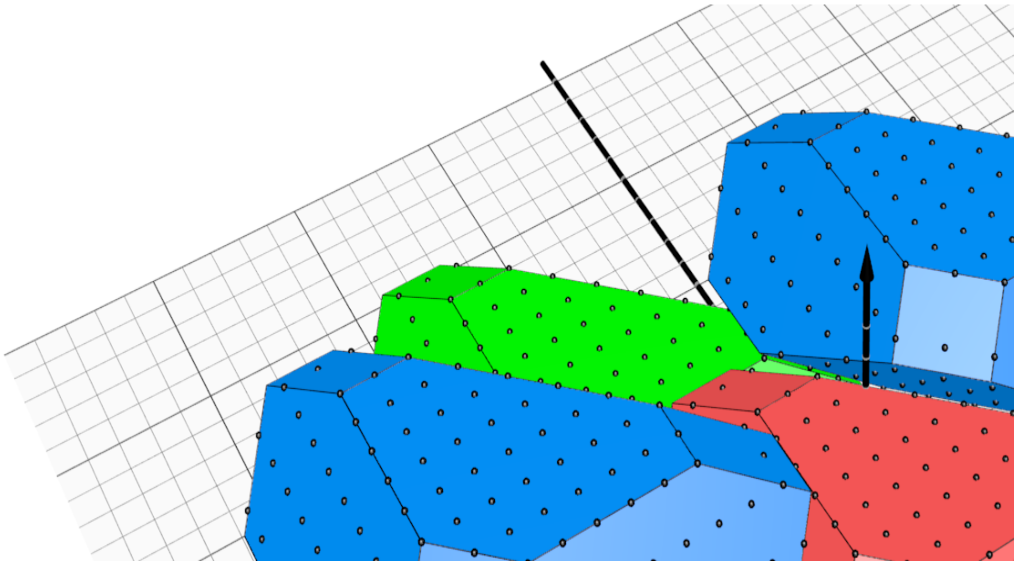
Technische Universiteit Eindhoven University of Technology

The polyhedral model

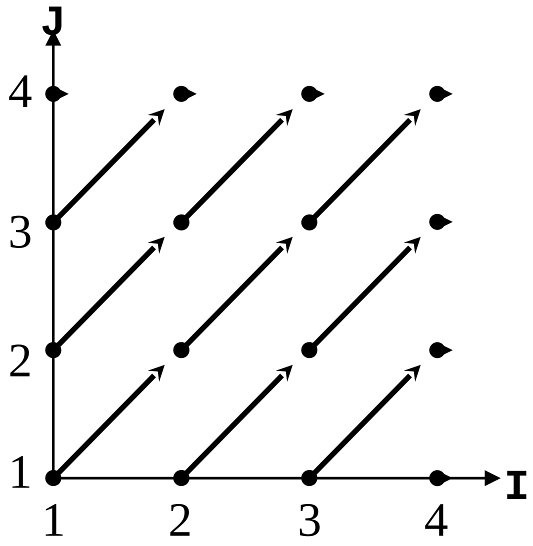
Roel Jordans



Introduction

- Remember dependency checking and the iteration domain?

```
    DO I = 1, 4
      DO J = 1, 4
S1      A(I,J+1) = A(I-1,J)
      ENDDO
    ENDDO
```



What can we do with it?

- Used for optimization passes
  - Each pass checks the **legality**
  - Analyzes **profitability**
  - Applies the **transformation**
- Things like
  - Auto vectorization
  - Loop unrolling
- *Fast results for specialized optimizations*

But in which order?

- Optimizations can enable other optimizations
- Iterative optimization
  - Finding the right sequence of optimizations
  - **Huge** exploration space
  - Extremely **slow**
  - Repeats the analysis for each step
- Can we combine steps?

```
for(i=0; i<3; ++i)
S:    B[i] = f(A[i]);
for(i=0; i<3; ++i)
T:    C[i] = g(B[2-i]);
```

{S[i]}, {T[i]}

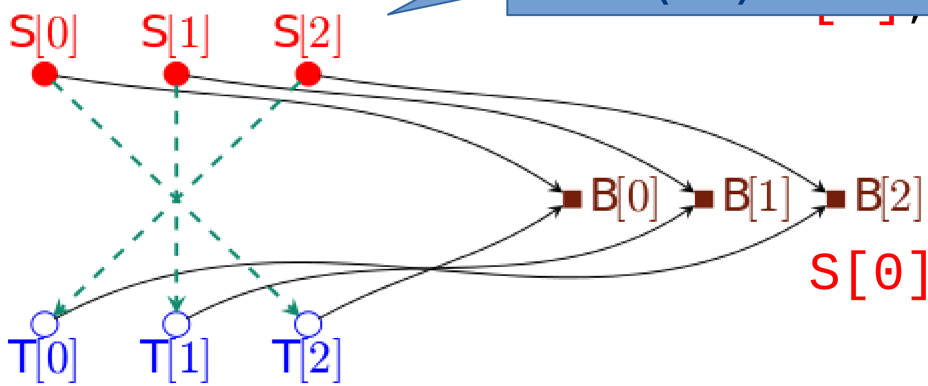
Only few are legal!

{S[i]-> [i]} {T[i]-> [i]}

6 statements
720 (6!) orderings

S[], T[]

S[0],S[1],S[2] T[0],T[1],T[2]



S[0],T[2],S[1],T[1],S[2],T[0]

S[], T[]

```
for(c=0; c<3; ++c){
    B[c] = f(A[c]);
    C[2-c] = g(B[c]);
}
```

{S[i]-> [i]};{T[i]-> [**2-i**]}

{S[i]}, {T[i]}

The polyhedral model

- Focus on Static Control Parts (**ScoP**)
- The iteration space is usually quite regular
  - We can define an access function!
  - But arbitrary access functions may be a bit much
  - Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques on the model

Advantages

- In the polyhedral model *(Feautrier, '92)*
  - Compositions of transformations are easily expressed
  - Transformation legality is easily checked
  - Natural expression of parallelism

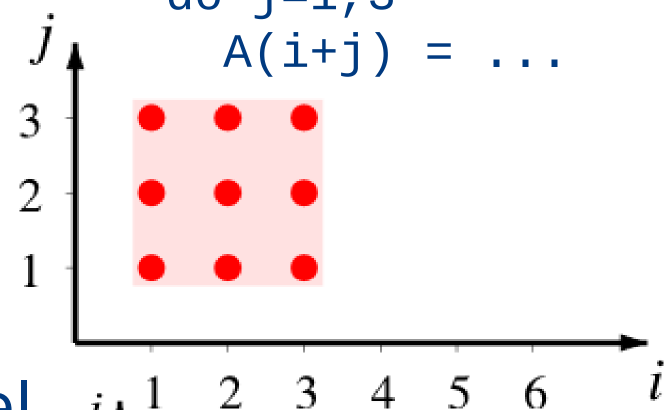
ScoP restrictions

- Static control: *Control flow within the loop only depends on the loop iterators*
  - `if(A[i] == X) …;` ← **not allowed**
  - `for(i=max(1,t-3); i < max(t-1, 3); i++)` ← **allowed**
- Loops with (quasi-)affine access functions allow dense storage
  - Only `a*i+b*j*c*k` (with a, b, c integers)
  - No `i*i`, `abs(i)`, …

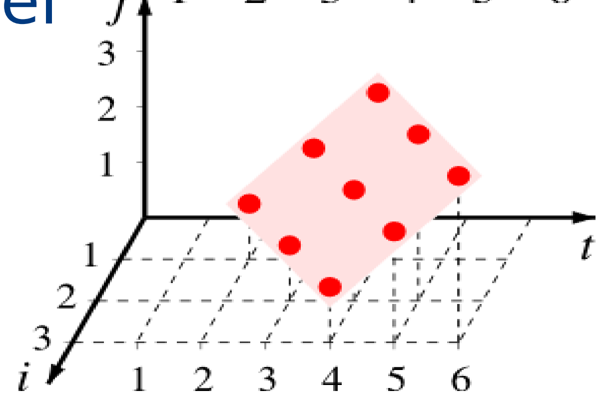
A three stage process

```
do i=1,3
  do j=1,3
    A(i+j) = ...
```

1 Analysis: from code to model



2 Transformation in the model
Here: Θ(i,j) = t = i+j



3 Code generation:
from model to code

```
do t=2,6
  do i=max(1,t-3), min(t-1,3)
    A(t) = ...
```

Extract the *Instance Set*

```
    do i=0,n
R      s(i) = 0
       do j=0,n
S         s(i) = s(i) + a(i,j)*x(j)
       end do
    end do
```

Extract the *Instance Set*

```
    do i=0,n
R       s(i) = 0
        do j=0,n
S           s(i) = s(i) + a(i,j)*x(j)
        end do
    end do
```

Iteration domain of *R*

- Iteration vector: $\mathbf{x}_R=(i)$
- Exact set of **instances** of R: $D_R : \{i \mid 0 <= i <= n\}$

Extract the *Instance Set*

```
    do i=0,n
R      s(i) = 0
       do j=0,n
S         s(i) = s(i) + a(i,j)*x(j)
       end do
    end do
```

Iteration domain of *S*

- Iteration vector: $\mathbf{x}_s=(i,j)$
- Exact set of **instances** of S:

  D_s : {i,j | 0 <= i <= n, 0 <= j <= n}

Scheduling a program

A schedule of a program is a function which associates a logical date (a timestamp) to each instance of each statement.

$$\Theta_S\left(\vec{x}_S\right) = T\begin{pmatrix}\vec{x}_S \\ \vec{n} \\ 1\end{pmatrix}$$

- Two instances having the same date can run in parallel
- Schedule dimensions corresponds to the number of nested sequential loops

Program transformations

- Every composition of loop transformations can be expressed as affine schedules *(Wolf, '92)*

- A schedule is the result of an **arbitrary complex composition** of transformations

A scheduling example

```
do i = 1,2
  do j=1,3
    a(i,j) = a(i,j) * 0.2
```

Original loop

$$\Theta_R\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{pmatrix} i \\ j \end{pmatrix}$$

```
do i = 1,2
  do j=1,3
    a(i,j) = a(i,j) * 0.2
```

A scheduling example

```
do i = 1,2
  do j=1,3
    a(i,j) = a(i,j) * 0.2
```

Loop interchange

$$\Theta_R \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} j \\ i \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

```
do j=1,3
  do i = 1,2
    a(i,j) = a(i,j) * 0.2
```

Finding legal schedules

Build the set of all *legal* program versions (i.e. which respects all the data dependence in the program)

- Perform an exact dependence analysis
- Build the set of all possible values of T

- The resulting space represents all the distinct possible ways to **legally reschedule** the program, using arbitrarily complex sequences of transformations

Dependence expression

- Need to represent the exact set of instances in dependence
- Exact computation made possible thanks to the ScoP and Static reference assumptions
- Use a subset of the Cartesian product of iteration domains

```
do i=0,n
   s(i) = 0
   do j=0,n
      s(i) = s(i) + a(i,j)*x(j)
   end do
end do
```

Dependence expression

```
    do i=1,n
R       s(i) = 0
        do j=1,n
S           s(i) = s(i) + a(i,j)*x(j)
        end do
    end do
```

Iterations of R

$$D_{R\delta S}: \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 \\ & & & & \\ & & & & \end{bmatrix} \cdot \begin{pmatrix} i_R \\ i_S \\ j_S \\ n \\ 1 \end{pmatrix}$$

Dependence expression

```
    do i=0,n
R       s(i) = 0
        do j=0,n
S           s(i) = s(i) + a(i,j)*x(j)
        end do
    end do
```

Iterations of S

$$D_{R\delta S}: \begin{bmatrix} 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} i_R \\ i_S \\ j_S \\ n \\ 1 \end{pmatrix}$$

Dependence expression

```
    do i=0,n
R      s(i) = 0
       do j=0,n
S         s(i) = s(i) + a(i,j)*x(j)
       end do
    end do
```

S depends on R

$$D_{R\delta S}: \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} i_R \\ i_S \\ j_S \\ n \\ 1 \end{pmatrix}$$

Dependence expression

```
     do i=0,n
R       s(i) = 0
        do j=0,n
S          s(i) = s(i) + a(i,j)*x(j)
        end do
     end do
```

Complete result

$$D_{R\delta S}: \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} i_R \\ i_S \\ j_S \\ n \\ 1 \end{pmatrix}$$

Definition: Legal schedule

Assuming RδS, schedules $\Theta_R(\mathbf{x}_R)$ and $\Theta_S(\mathbf{x}_S)$ are legal iff:

$$\Delta_{R,S} = \Theta_S(\vec{x}_S) - \Theta_R(\vec{x}_R) - 1$$

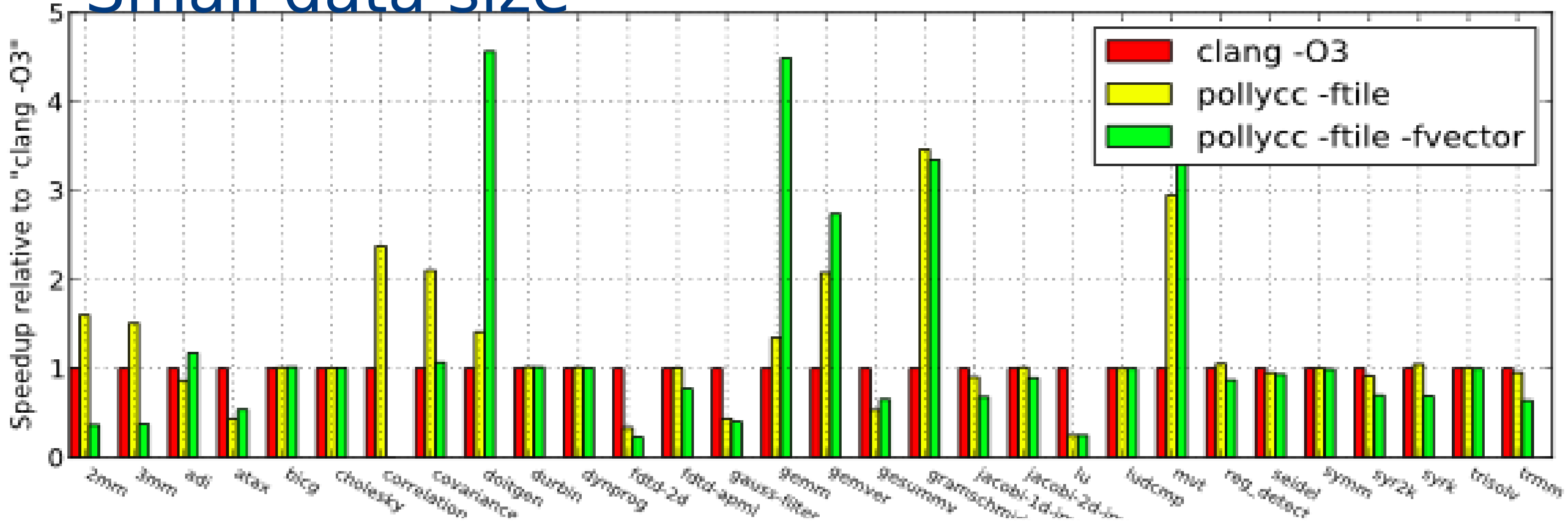
Is non-negative for each point in $D_{r\delta S}$

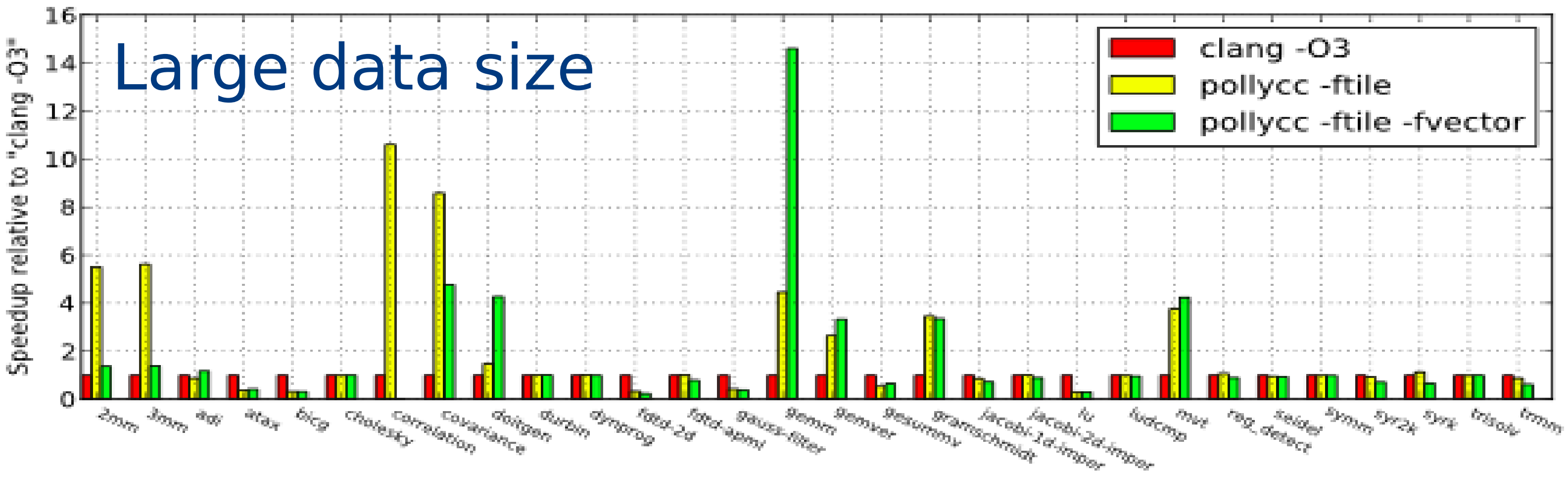
Polyhedral tools

- PET *(Polyhedral Extraction Tool)*
  - Analysis only
  - Based on clang AST
- Polly
  - Analyzes and transforms LLVM IR
- GRAPHITE
  - Like Polly but works on GCC's GIMPLE
- ISL *(Integer Set Library)*
  - Set calculus for polyhedral analysis
  - Used by PET and Polly

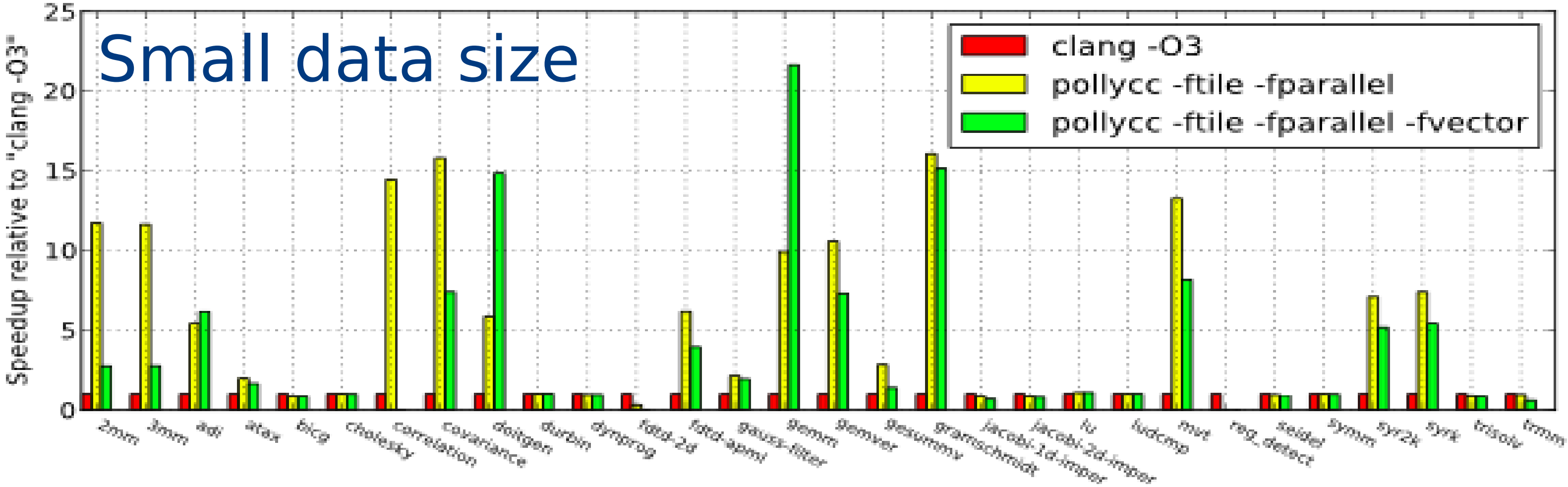
Effects 1 (2011): Sequential

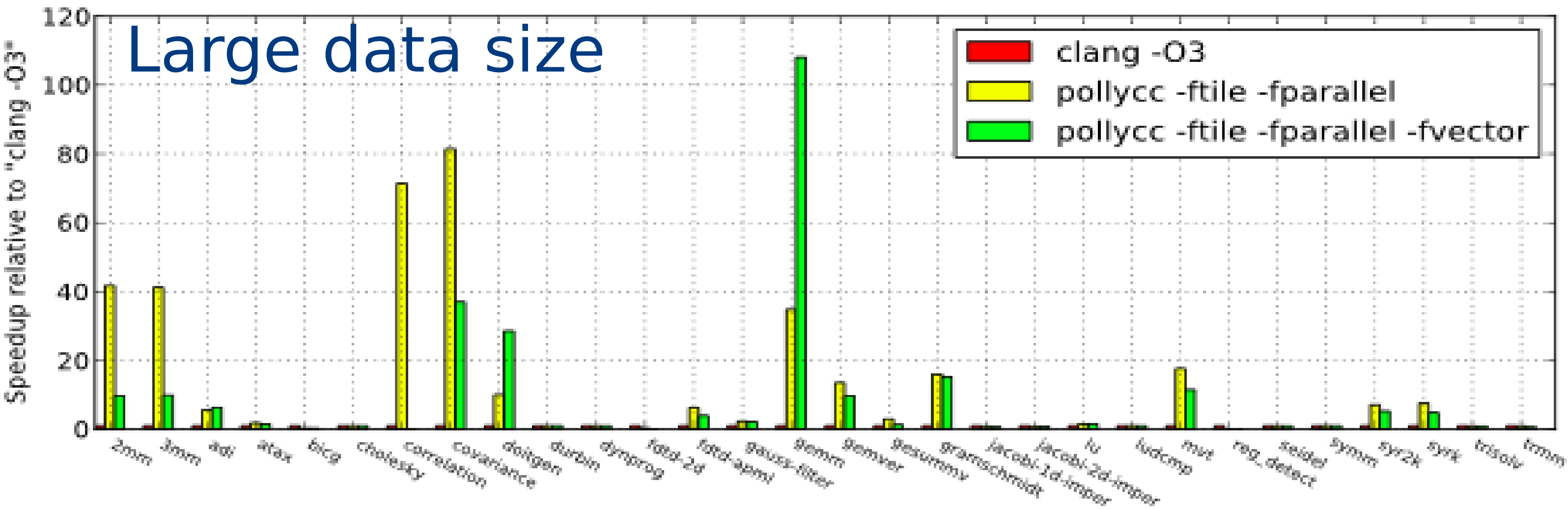
Small data size





Effects 2: Parallel (24 thread)





Useful material

Tobias Grosser & Johannes Doerfert (2015) – ***Polly: Optimistic Loop Nest Optimizations with Schedule Trees***

- Tutorial session from the LLVM developers' meeting
- https://www.youtube.com/watch?v=mIBUY20d8c8

Sven Verdoolage (2016) – ***Presburger Formulas and Polyhedral Compilation***

- Tutorial style book with all the details if you really want to dive into this
- *Check chapter 5 for more info on today's lecture*

Louis-Noel Pouchet, *et al.* (2006) – ***Iterative Optimization in the Polyhedral Model: One-Dimensional Affine Schedules***

- More of the proofs presented today