Tutorial on Lambda and Graphite Loop Transformations

Sebastian Pop

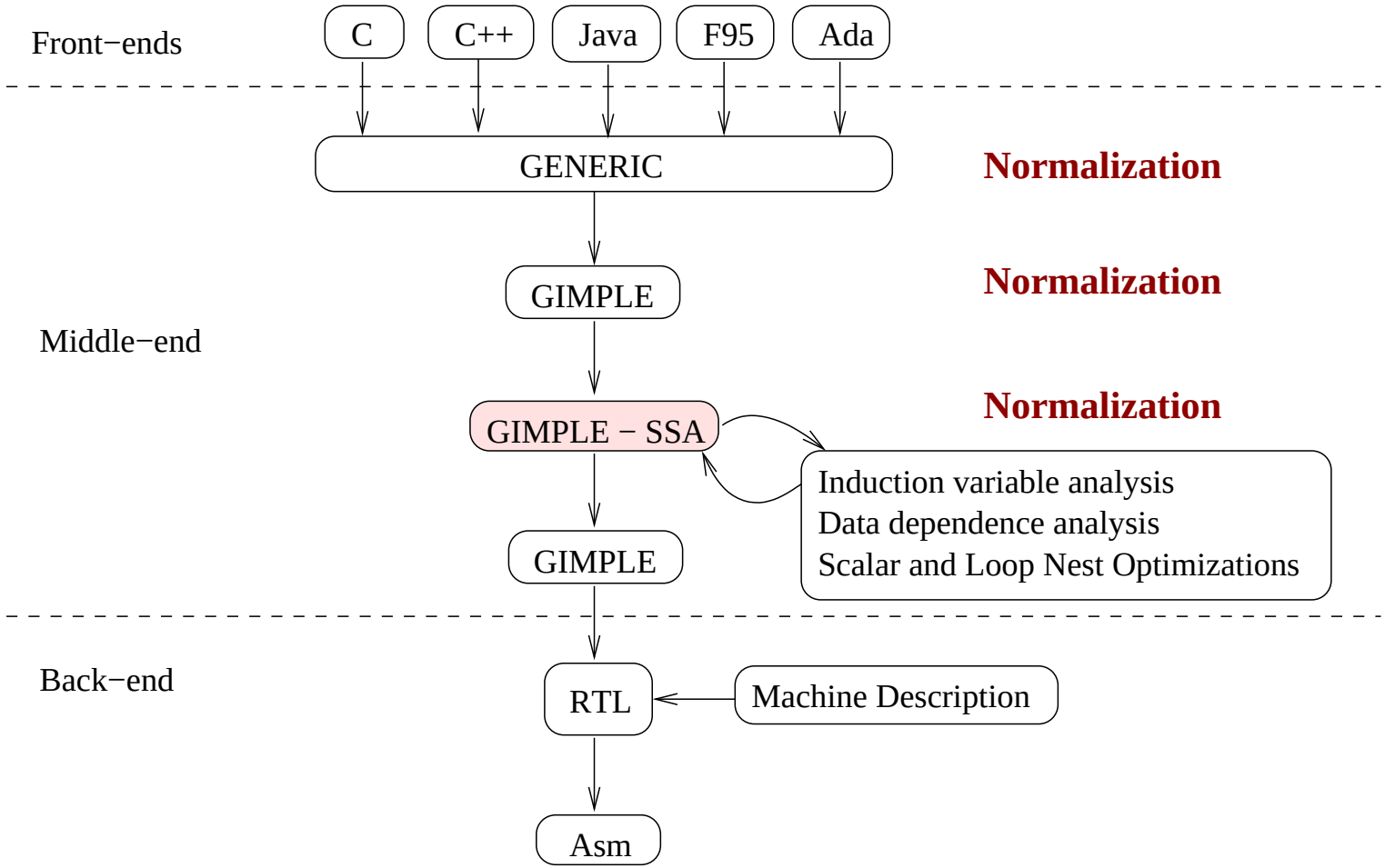
AMD - Austin, Texas

GCC Summit 2008,
June 18, 2008

Outline

- current loop transforms in GCC
- polyhedral model (Graphite and Lambda)
- selecting loop transforms for parallel, vector, or stream code

Where are the Loop Transforms?



Compilation to Polyhedra (Linear Constraints)

toPoly translates imperative language constructs to declarations

- loops $\longrightarrow$ { iteration domains, dynamic scheduling functions }
- arbitrary statement sequence $\longrightarrow$ static scheduling functions
- array references $\longrightarrow$ memory access functions
- reaching definitions $\longrightarrow$ data dependences

also see

- static single assignment (SSA) [CFRWZ'91]
- systems of uniform recurrence equations (SURE) [KMW'67]

Polyhedral Representation

1. iteration domain = bounds of enclosing loops

```
for (i=0; i<m; i++)
 for (j=5; j<n; j++)
  A[2*i][j+1] = ...
```

$$\left[\begin{array}{ccccc|c} i & j & m & n & cst & \\ \hline 1 & 0 & 0 & 0 & 0 & i \geq 0 \\ -1 & 0 & 1 & 0 & -1 & -i + m \geq -1 \\ 0 & 1 & 0 & 0 & 5 & j \geq 5 \\ 0 & -1 & 0 & 1 & -1 & -j + n \geq -1 \end{array}\right]$$

Polyhedral Representation

1. iteration domain = bounds of enclosing loops
2. schedule = execution time (static + dynamic)

- sequence $[\![s_1; s_2]\!]$:
  $\mathcal{S}[\![s_1]\!] = \mathrm{t}, \qquad \mathcal{S}[\![s_2]\!] = \mathrm{t} + 1$
- loop $[\![loop_1\ s\ end_1]\!]$: i_1 indexes $loop_1$ iterations: dynamic time
  $\mathcal{S}[\![loop_1]\!] = \mathrm{t}, \qquad \mathcal{S}[\![s]\!] = (\mathrm{t}, i_1, 0)$

Polyhedral Representation

1. iteration domain = bounds of enclosing loops
2. schedule = execution time (static + dynamic)
3. access functions

```
for (i=0; i<m; i++)
 for (j=5; j<n; j++)
  A[2*i][j+1] = ...
```

$$\left[\begin{array}{ccccc|c} i & j & m & n & cst & \\ \hline 2 & 0 & 0 & 0 & 0 & 2*i \\ 0 & 1 & 0 & 0 & 1 & j+1 \end{array}\right]$$

Transformations in the Polyhedral Model

Pugh'91, a loop transform is a combination of:

- loop interchange
- loop skewing
- loop reversal
- loop blocking
- loop fusion
- loop distribution
- statement reordering

in the polyhedral model loop transforms have a low cost: operations on matrices

Analyses in the Polyhedral Model

- Polyhedra is a complete representation of the Imp program
- analyses can be performed uniquely on Polyhedra

Counting Points in Polyhedra

In many program analyses and optimisations, questions starting with "how many" need to be answered:

- How many memory locations are touched by a loop?
- How many operations are performed by a loop?
- How many cache lines are touched by a loop?
- How many array elements are accessed between two points?
- How many array elements are live at a given iteration?
- How many times is a statement executed before an iteration?
- How many cache misses does a loop generate?
- How much memory is dynamically allocated?

Techniques used for counting points:

- Ehrhart polynomials
- Barvinok's generating functions

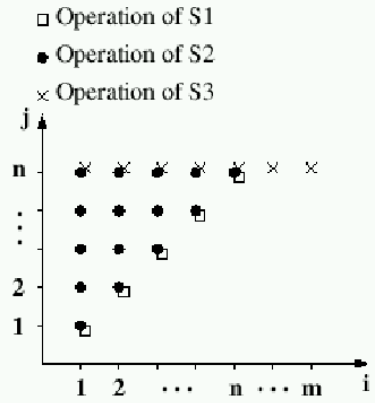
Data Dependence Analysis

- data dependences characterise computation sharing
- sharing $\Rightarrow$ synchronisation and communications
- no sharing $=$ parallelism $=$ recomputations
- transform legality $=$ satisfying original computation order

Imp Generation from Polyhedra

toImp introduces imperative language constructs: sequence, loops, parallel computations, communication, ...

CLooG's Imp Generation from Polyhedra



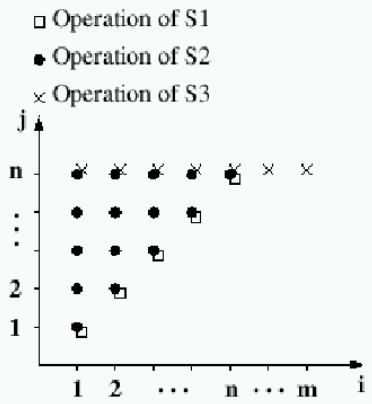
$$\mathcal{T}_{S_1} : \begin{cases} 1 \leq i \leq n \\ j = i \end{cases}$$

$$\mathcal{T}_{S_2} : \begin{cases} 1 \leq i \leq n \\ i \leq j \leq n \end{cases}$$

$$\mathcal{T}_{S_3} : \begin{cases} 1 \leq i \leq m \\ j = n \end{cases}$$

(a) Initial domains to scan

CLooG's Imp Generation from Polyhedra

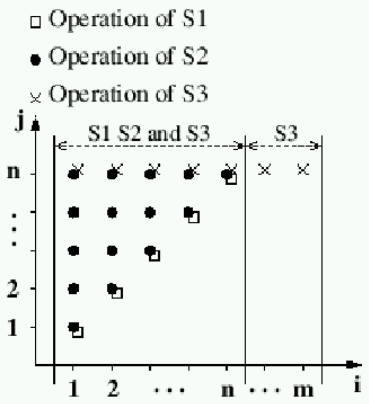


$$\mathcal{T}_{S_1} : \begin{cases} 1 \leq i \leq n \\ j = i \end{cases}$$

$$\mathcal{T}_{S_2} : \begin{cases} 1 \leq i \leq n \\ i \leq j \leq n \end{cases}$$

$$\mathcal{T}_{S_3} : \begin{cases} 1 \leq i \leq m \\ j = n \end{cases}$$

(a) Initial domains to scan




```
do i=1, n
```

$$\mathcal{T}_{S_1} : \begin{cases} 1 \leq i \leq n \\ j = i \end{cases}$$

$$\mathcal{T}_{S_2} : \begin{cases} 1 \leq i \leq n \\ i \leq j \leq n \end{cases}$$

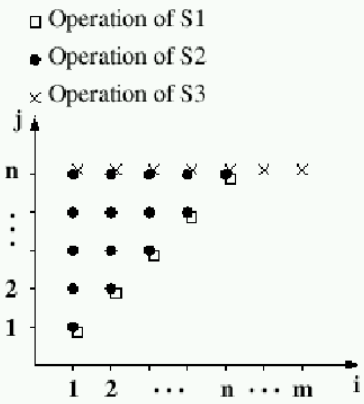
$$\mathcal{T}_{S_3} : \begin{cases} 1 \leq i \leq n \\ j = n \end{cases}$$

```
do i=n+1, m
```

$$\mathcal{T}_{S_3} : \begin{cases} n+1 \leq i \leq m \\ j = n \end{cases}$$

(b) Projection and separation onto the first dimension

CLooG's Imp Generation from Polyhedra

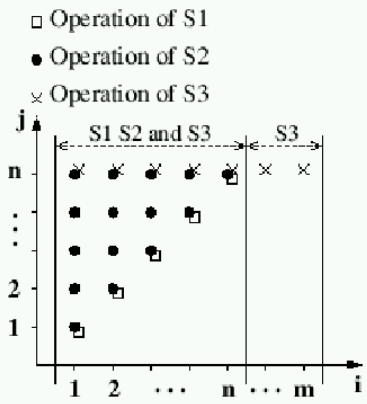


$$\mathcal{T}_{S_1} : \begin{cases} 1 \le i \le n \\ j = i \end{cases}$$

$$\mathcal{T}_{S_2} : \begin{cases} 1 \le i \le n \\ i \le j \le n \end{cases}$$

$$\mathcal{T}_{S_3} : \begin{cases} 1 \le i \le m \\ j = n \end{cases}$$

(a) Initial domains to scan




```
do i=1, n
```

$$\mathcal{T}_{S_1} : \begin{cases} 1 \le i \le n \\ j = i \end{cases}$$

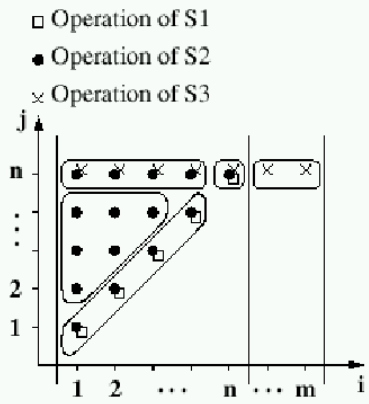
$$\mathcal{T}_{S_2} : \begin{cases} 1 \le i \le n \\ i \le j \le n \end{cases}$$

$$\mathcal{T}_{S_3} : \begin{cases} 1 \le i \le n \\ j = n \end{cases}$$

```
do i=n+1, m
```

$$\mathcal{T}_{S_3} : \begin{cases} n+1 \le i \le m \\ j = n \end{cases}$$

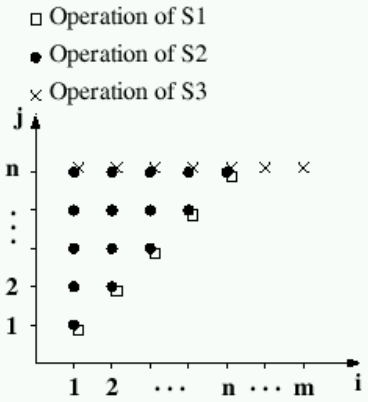
(b) Projection and separation onto the first dimension




```
do i=1, n
|  if (i==n) then
|  |  S1(j=n)
|  |  S2(j=n)
|  |  S3(j=n)
|  if (i<=n-1) then
|  |  S1(j=i)
|  |  S2(j=i)
|  do j=i+1, n-1
|  |  S2
|  if (i<=n-1) then
|  |  S2(j=n)
|  |  S3(j=n)
do i=n+1, m
|  S3(j=n)
```

(c) Recursion on next dimension

CLooG's Imp Generation from Polyhedra

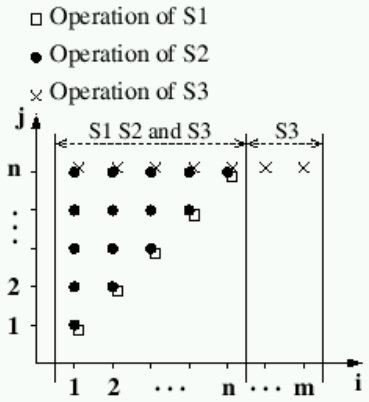


$$\mathcal{T}_{S_1} : \begin{cases} 1 \le i \le n \\ j = i \end{cases}$$

$$\mathcal{T}_{S_2} : \begin{cases} 1 \le i \le n \\ i \le j \le n \end{cases}$$

$$\mathcal{T}_{S_3} : \begin{cases} 1 \le i \le m \\ j = n \end{cases}$$

(a) Initial domains to scan




```
do i=1, n
```

$$\mathcal{T}_{S_1} : \begin{cases} 1 \le i \le n \\ j = i \end{cases}$$

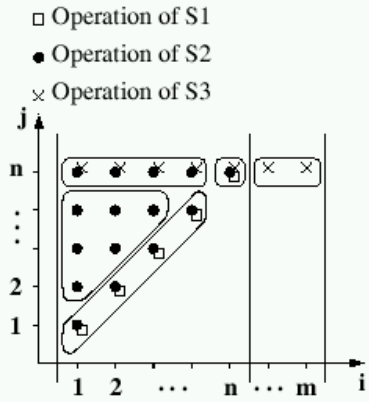
$$\mathcal{T}_{S_2} : \begin{cases} 1 \le i \le n \\ i \le j \le n \end{cases}$$

$$\mathcal{T}_{S_3} : \begin{cases} 1 \le i \le n \\ j = n \end{cases}$$

```
do i=n+1, m
```

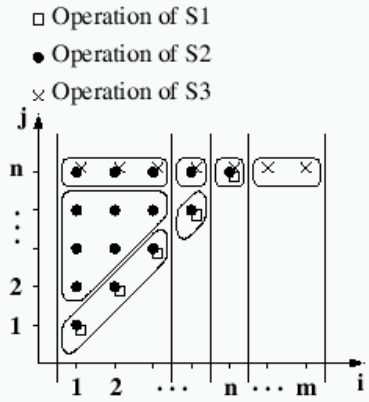
$$\mathcal{T}_{S_3} : \begin{cases} n+1 \le i \le m \\ j = n \end{cases}$$

(b) Projection and separation onto the first dimension




```
do i=1, n
  if (i==n) then
    S1(j=n)
    S2(j=n)
    S3(j=n)
  if (i<=n-1) then
    S1(j=i)
    S2(j=i)
  do j=i+1, n-1
    S2
  if (i<=n-1) then
    S2(j=n)
    S3(j=n)
do i=n+1, m
  S3(j=n)
```

(c) Recursion on next dimension




```
do i=1, n-2
  S1(j=i)
  S2(j=i)
  do j=i+1, n-1
    S2
  S2(j=n)
  S3(j=n)
S1(i=n-1,j=n-1)
S2(i=n-1,j=n-1)
S2(i=n-1,j=n)
S3(i=n-1,j=n)
S1(i=n,j=n)
S2(i=n,j=n)
S3(i=n,j=n)
do i=n+1, m
  S3(j=n)
```

(d) Backtrack with dead code removing

Loopo's Imp Generation from Polyhedra

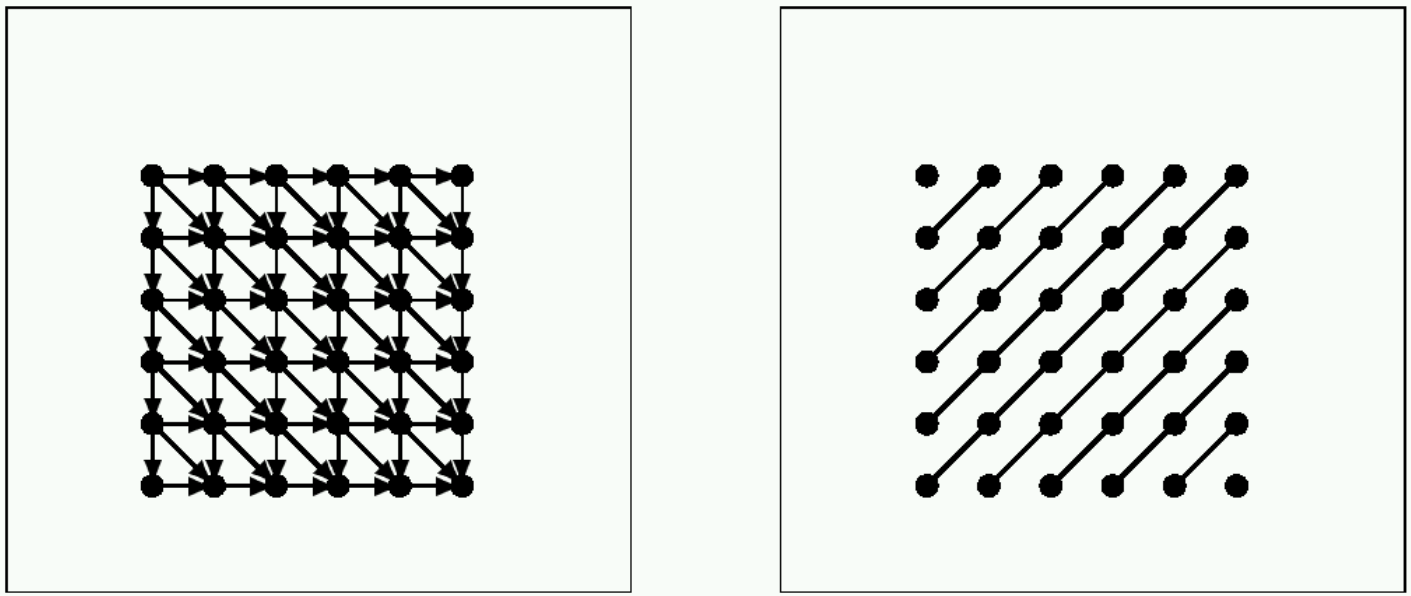
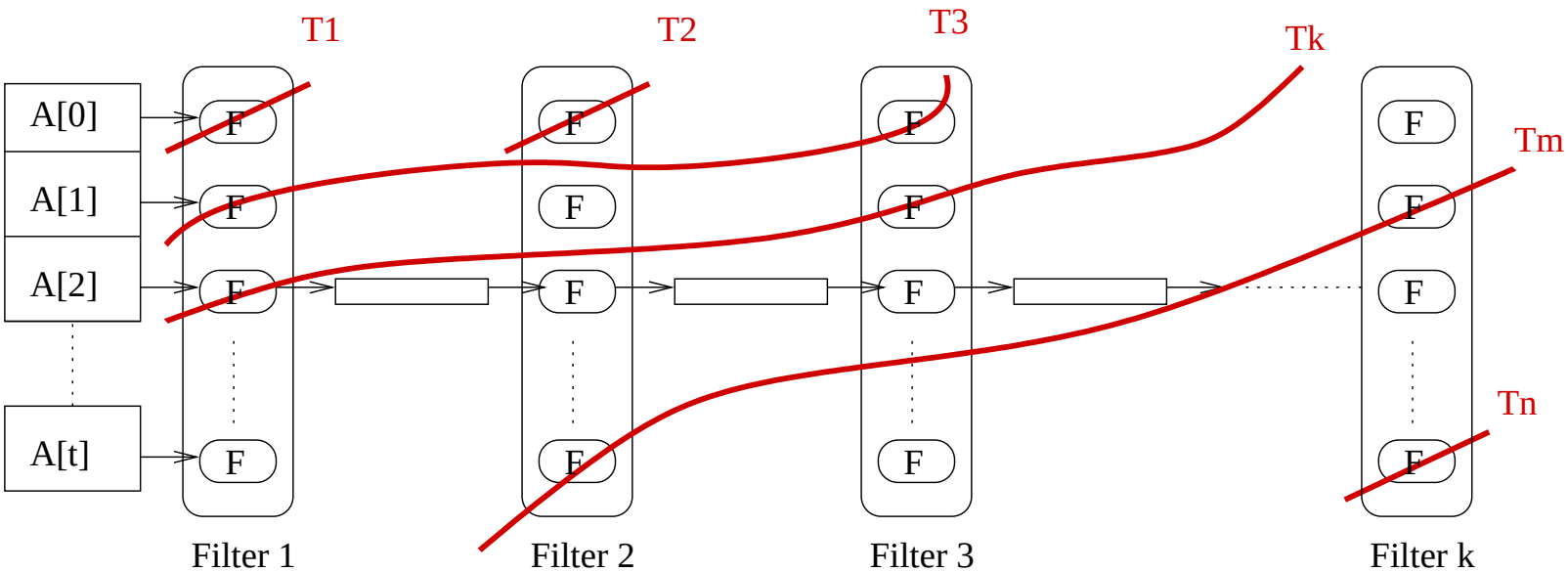


Fig. 5. The dependence graph of the uniform recurrence equations (left) and the time slices of a minimal affine schedule (right).

Stream Code Generation from Polyhedra



How to Select Loop Transforms?

- loop transform selection is still an open question
- in function of the code generation strategy

Conclusion

- Current polyhedral representations mix imperative constructs (assignments, statements, sequences) to higher level declarative constructs (iteration domains, access functions)
- out-of-Polyhedra compilers introduce imperative constructs: sequence, sharing, parallel operations, ...
- Hard part: decide on the amount of sharing computations based on the characteristics of the machine: bandwidth, latency, speed of processors, load of computations, ...