Schedule Trees

Sven Verdoolaege[1] Serge Guelton[2]
Tobias Grosser[3] Albert Cohen[3]

[1] INRIA, École Normale Supérieure and KU Leuven

[2] École Normale Supérieure and Télécom Bretagne

[3] INRIA and École Normale Supérieure

January 20, 2014

Outline

1. Introduction
   - Example
   - Single Statement
   - Multiple Statements
   - Schedule Trees
2. Advantages
   - Useful in several contexts
   - More natural
   - More convenient
   - More expressive
   - Extensible
3. Conclusion

Outline

1. Introduction
    - Example
    - Single Statement
    - Multiple Statements
    - Schedule Trees
2. Advantages
    - Useful in several contexts
    - More natural
    - More convenient
    - More expressive
    - Extensible
3. Conclusion

Introductory Example

```
for (i = 0; i <= N; ++i)
S:   a[i] = g(i);
for (i = 0; i <= N; ++i)
T:   b[i] = f(a[N-i]);
```

Introductory Example

```
for (i = 0; i <= N; ++i)
S:   a[i] = g(i);
for (i = 0; i <= N; ++i)
T:   b[i] = f(a[N-i]);
```

- Iteration domain

$$\{\, \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N \,\}$$

- Dependences

$$\{\, \mathtt{S}[i] \to \mathtt{T}[N-i] : 0 \le i \le N \,\}$$

- Execution Order
  - Original Order

$$\mathtt{S}[0], \mathtt{S}[1], \mathtt{S}[2], \ldots, \mathtt{S}[N-1], \mathtt{S}[N], \mathtt{T}[0], \mathtt{T}[1], \mathtt{T}[2], \ldots, \mathtt{T}[N-1], \mathtt{T}[N]$$

Introductory Example

```
for (i = 0; i <= N; ++i)
S:  a[i] = g(i);
for (i = 0; i <= N; ++i)
T:  b[i] = f(a[N-i]);
```

- Iteration domain

$$\{\, \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N \,\}$$

- Dependences

$$\{\, \mathtt{S}[i] \to \mathtt{T}[N-i] : 0 \le i \le N \,\}$$

- Execution Order
  - Original Order

$$\mathtt{S}[0], \mathtt{S}[1], \mathtt{S}[2], \ldots, \mathtt{S}[N-1], \mathtt{S}[N], \mathtt{T}[0], \mathtt{T}[1], \mathtt{T}[2], \ldots, \mathtt{T}[N-1], \mathtt{T}[N]$$

  - Alternative Order

$$\mathtt{S}[0], \mathtt{T}[N], \mathtt{S}[1], \mathtt{T}[N-1], \mathtt{S}[2], \mathtt{T}[N-2], \ldots, \mathtt{S}[N-1], \mathtt{T}[1], \mathtt{S}[N], \mathtt{T}[0]$$

Introductory Example

```
for (i = 0; i <= N; ++i)
S:  a[i] = g(i);
for (i = 0; i <= N; ++i)
T:  b[i] = f(a[N-i]);
```

```
for (i = 0; i <= N; ++i) {
    a[i] = g(i);
    b[N-i] = f(a[i]);
}
```

- Iteration domain

$$\{\, \mathtt{S}[i] : 0 \leq i \leq N; \mathtt{T}[i] : 0 \leq i \leq N \,\}$$

- Dependences

$$\{\, \mathtt{S}[i] \rightarrow \mathtt{T}[N - i] : 0 \leq i \leq N \,\}$$

- Execution Order
  - Original Order

$$\mathtt{S}[0], \mathtt{S}[1], \mathtt{S}[2], \ldots, \mathtt{S}[N-1], \mathtt{S}[N], \mathtt{T}[0], \mathtt{T}[1], \mathtt{T}[2], \ldots, \mathtt{T}[N-1], \mathtt{T}[N]$$

  - Alternative Order

$$\mathtt{S}[0], \mathtt{T}[N], \mathtt{S}[1], \mathtt{T}[N-1], \mathtt{S}[2], \mathtt{T}[N-2], \ldots, \mathtt{S}[N-1], \mathtt{T}[1], \mathtt{S}[N], \mathtt{T}[0]$$

Expressing Transformations (Single Statement)

```
for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);
```

⇒

```
for (i = 0; i <= N; ++i)
    b[N-i] = f(a[i]);
```

Expressing Transformations (Single Statement)

```
for (i = 0; i <= N; ++i)              for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);          ⇒          b[N-i] = f(a[i]);
```

Two approaches

1. Modify Iteration Domain

$$\mathtt{T}[i] \rightarrow \mathtt{T}'[N-i]$$

    - iteration domains have implicit execution order (lexicographic order)
    - AST generator takes modified iteration domain as input
    - access relations and dependence relations are adjusted accordingly

Expressing Transformations (Single Statement)

```
for (i = 0; i <= N; ++i)        ⇒   for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);                   b[N-i] = f(a[i]);
```

Two approaches

1. Modify Iteration Domain

$$\mathtt{T}[i] \rightarrow \mathtt{T}'[N - i]$$

    - iteration domains have implicit execution order (lexicographic order)
    - AST generator takes modified iteration domain as input
    - access relations and dependence relations are adjusted accordingly

2. Explicit Schedule

$$\mathtt{T}[i] \rightarrow [N - i]$$

    - iteration domains have no implicit execution order
    - execution order is determined by schedule space (lexicographic order)
    - AST generator takes iteration domain and schedule as input
    - schedule is typically a piecewise quasi-affine function

Expressing Transformations (Single Statement)

```
for (i = 0; i <= N; ++i)           for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);          ⇒        b[N-i] = f(a[i]);
```

Two approaches

1. Modify Iteration Domain

$$\mathtt{T}[i] \rightarrow \mathtt{T}'[N-i]$$

   - iteration domains have implicit execution order (lexicographic order)
   - AST generator takes modified iteration domain as input
   - access relations and dependence relations are adjusted accordingly

2. Explicit Schedule

$$\mathtt{T}[i] \rightarrow [N-i]$$

   - iteration domains have no implicit execution order
   - execution order is determined by schedule space (lexicographic order)
   - AST generator takes iteration domain and schedule as input
   - schedule is typically a piecewise quasi-affine function

Expressing Transformations (Single Statement)

```
for (i = 0; i <= N; ++i)          for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);        ⇒        b[N-i] = f(a[i]);
```

Two approaches

1. Modify Iteration Domain

$$\mathtt{T}[i] \to \mathtt{T}'[N-i]$$

    - iteration domains have implicit execution order (lexicographic order)
    - AST generator takes modified iteration domain as input
    - access relations and dependence relations are adjusted accordingly

2. Explicit Schedule

$$\mathtt{T}[i] \to [N-i]$$

    - iteration domains have no implicit execution order
    - execution order is determined by schedule space (lexicographic order)
    - AST generator takes iteration domain and schedule as input
    - schedule is typically a piecewise quasi-affine function

Representing Schedules for Multiple Statements

```
for (i = 0; i <= N; ++i)
    a[i] = g(i);
for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);
```

```
for (i = 0; i <= N; ++i) {
    a[i] = g(i);
    b[N-i] = f(a[i]);
}
```

$S[i] \to [i]; T[i] \to [N - i]$

first $S[i]$ then $T[i]$

$S[i] \to [i]$ $T[i] \to [i]$

first $S[i]$ then $T[i]$

Representing Schedules for Multiple Statements

```
for (i = 0; i <= N; ++i)
    a[i] = g(i);
for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);
```

```
for (i = 0; i <= N; ++i) {
    a[i] = g(i);
    b[N-i] = f(a[i]);
}
```

$S[i] \to [i]; T[i] \to [N - i]$

first $S[i]$ then $T[i]$

$S[i] \to [i]$ $T[i] \to [i]$

first $S[i]$ then $T[i]$

Kelly

$S : \{ [i] \to [0, i] \}$

$T : \{ [i] \to [1, i] \}$

$S : \{ [i] \to [i, 0] \}$

$T : \{ [i] \to [N - i, 1] \}$

⇒ *encode* statement ordering in affine function

Representing Schedules for Multiple Statements

```
for (i = 0; i <= N; ++i)
    a[i] = g(i);
for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);
```

```
for (i = 0; i <= N; ++i) {
    a[i] = g(i);
    b[N-i] = f(a[i]);
}
```

$\mathtt{S}[i] \to [i]; \mathtt{T}[i] \to [N-i]$

first $\mathtt{S}[i]$ then $\mathtt{T}[i]$

$\mathtt{S}[i] \to [i]$ $\mathtt{T}[i] \to [i]$

first $\mathtt{S}[i]$ then $\mathtt{T}[i]$

Kelly

$\mathtt{S} : \{ [i] \to [0, i] \}$

$\mathtt{T} : \{ [i] \to [1, i] \}$

$\mathtt{S} : \{ [i] \to [i, 0] \}$

$\mathtt{T} : \{ [i] \to [N-i, 1] \}$

union map

$\{ \mathtt{S}[i] \to [0, i]; \mathtt{T}[i] \to [1, i] \}$

$\{ \mathtt{S}[i] \to [i, 0]; \mathtt{T}[i] \to [N-i, 1] \}$

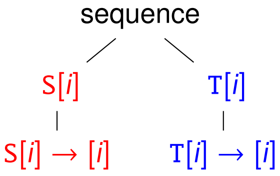
⇒ *encode* statement ordering in affine function

Representing Schedules for Multiple Statements

```
for (i = 0; i <= N; ++i)
    a[i] = g(i);
for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);
```

```
for (i = 0; i <= N; ++i) {
    a[i] = g(i);
    b[N-i] = f(a[i]);
}
```

schedule tree

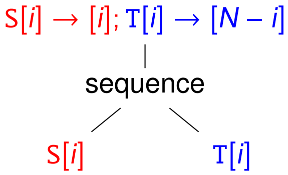

sequence

$\mathtt{S}[i]$

$\mathtt{T}[i]$

$\mathtt{S}[i] \to [i]$

$\mathtt{T}[i] \to [i]$


$\mathtt{S}[i] \to [i]; \mathtt{T}[i] \to [N - i]$

sequence

$\mathtt{S}[i]$

$\mathtt{T}[i]$

Kelly

$\mathtt{S} : \{ [i] \to [0, i] \}$

$\mathtt{T} : \{ [i] \to [1, i] \}$

$\mathtt{S} : \{ [i] \to [i, 0] \}$

$\mathtt{T} : \{ [i] \to [N - i, 1] \}$

union map

$\{ \mathtt{S}[i] \to [0, i]; \mathtt{T}[i] \to [1, i] \}$

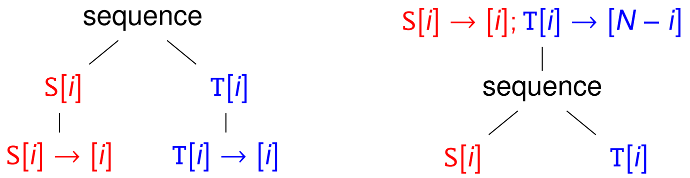
$\{ \mathtt{S}[i] \to [i, 0]; \mathtt{T}[i] \to [N - i, 1] \}$

Representing Schedules for Multiple Statements

```
for (i = 0; i <= N; ++i)
    a[i] = g(i);
for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i]);
```

```
for (i = 0; i <= N; ++i) {
    a[i] = g(i);
    b[N-i] = f(a[i]);
}
```

schedule tree



Kelly

$\mathrm{S} : \{\, [i] \to [0, i] \,\}$

$\mathrm{T} : \{\, [i] \to [1, i] \,\}$

$\mathrm{S} : \{\, [i] \to [i, 0] \,\}$

$\mathrm{T} : \{\, [i] \to [N - i, 1] \,\}$

union map

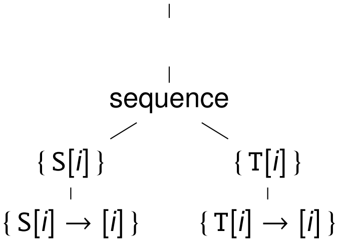
$\{\, \mathrm{S}[i] \to [0, i]; \mathrm{T}[i] \to [1, i] \,\}$

$\{\, \mathrm{S}[i] \to [i, 0]; \mathrm{T}[i] \to [N - i, 1] \,\}$

Other representations:

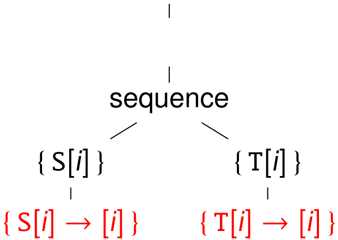
- "$2d + 1$": special case of Kelly's abstraction
- band forest: precursor to schedule trees

Schedule Trees



- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order

Schedule Trees


sequence

$\{ S[i] \}$

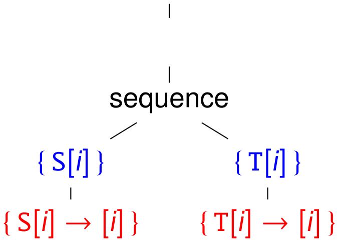
$\{ T[i] \}$

$\{ S[i] \to [i] \}$

$\{ T[i] \to [i] \}$

- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order

Schedule Trees



sequence

$\{ \mathtt{S}[i] \}$

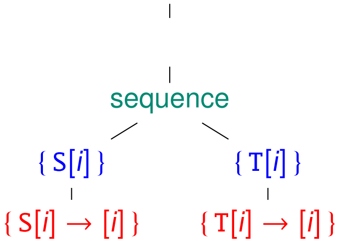
$\{ \mathtt{T}[i] \}$

$\{ \mathtt{S}[i] \rightarrow [i] \}$

$\{ \mathtt{T}[i] \rightarrow [i] \}$

- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order

Schedule Trees



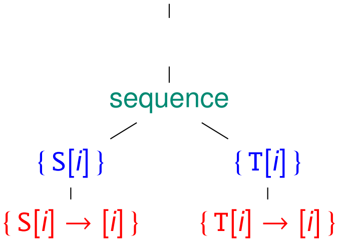
sequence

$\{ \mathtt{S}[i] \}$ $\{ \mathtt{T}[i] \}$

$\{ \mathtt{S}[i] \to [i] \}$ $\{ \mathtt{T}[i] \to [i] \}$

- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order

Schedule Trees



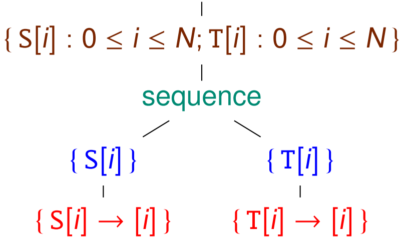
sequence

$\{ \mathtt{S}[i] \}$ $\{ \mathtt{T}[i] \}$

$\{ \mathtt{S}[i] \to [i] \}$ $\{ \mathtt{T}[i] \to [i] \}$

- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order
- “External” node types
  - Domain: set of statement instances to be scheduled
  - Context: external constraints on symbolic constants

Schedule Trees



$\{ \mathrm{S}[i] : 0 \le i \le N; \mathrm{T}[i] : 0 \le i \le N \}$

sequence

$\{ \mathrm{S}[i] \}$

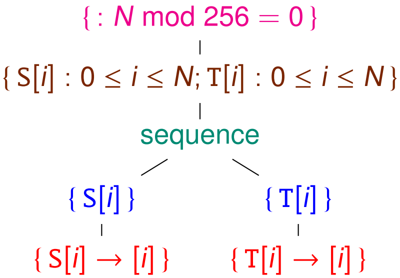
$\{ \mathrm{T}[i] \}$

$\{ \mathrm{S}[i] \to [i] \}$

$\{ \mathrm{T}[i] \to [i] \}$

- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order
- “External” node types
  - Domain: set of statement instances to be scheduled
  - Context: external constraints on symbolic constants

Schedule Trees



$\{ : N \bmod 256 = 0 \}$

$\{ \mathtt{S}[i] : 0 \leq i \leq N; \mathtt{T}[i] : 0 \leq i \leq N \}$

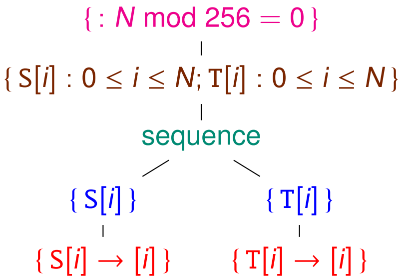
sequence

$\{ \mathtt{S}[i] \}$ $\{ \mathtt{T}[i] \}$

$\{ \mathtt{S}[i] \to [i] \}$ $\{ \mathtt{T}[i] \to [i] \}$

- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order
- “External” node types
  - Domain: set of statement instances to be scheduled
  - Context: external constraints on symbolic constants

Schedule Trees



$\{ : N \bmod 256 = 0 \}$

$\{ \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N \}$

sequence

$\{ \mathtt{S}[i] \}$ $\{ \mathtt{T}[i] \}$

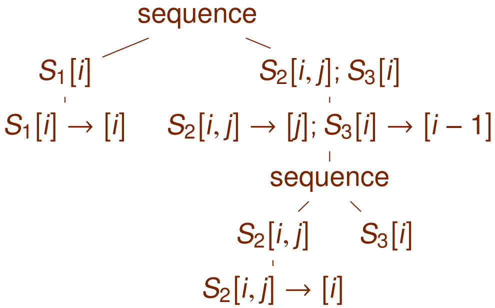
$\{ \mathtt{S}[i] \to [i] \}$ $\{ \mathtt{T}[i] \to [i] \}$

- Core node types
  - Band: multi-dimensional piecewise quasi-affine partial schedule
  - Filter: selects statement instances that are executed by descendants
  - Sequence: children executed in given order
  - Set: children executed in arbitrary order
- “External” node types
  - Domain: set of statement instances to be scheduled
  - Context: external constraints on symbolic constants
- Convenience node types
  - Mark: attach additional information to subtrees
  - Leaf: for easy navigation

Comparison

$$\begin{array}{lllll} T_1 : \{[i] & \to [0, i & & &]\} \\ T_2 : \{[i, j] & \to [1, j, & 0, i & &]\} \\ T_3 : \{[i] & \to [1, i-1, & 1 & &]\} \end{array}$$

$$\begin{array}{l} \{\, S_1[i] \to [0, i, 0, 0]; \\ \;\; S_2[i, j] \to [1, j, 0, i]; \\ \;\; S_3[i] \to [1, i-1, 1, 0] \,\} \end{array}$$



sequence

$S_1[i]$ $\quad$ $S_2[i, j]; S_3[i]$

$S_1[i] \to [i]$ $\quad$ $S_2[i, j] \to [j]; S_3[i] \to [i-1]$

sequence

$S_2[i, j]$ $\quad$ $S_3[i]$

$S_2[i, j] \to [i]$

- Kelly's abstraction
  - schedule spread over statements
  - relaxed lexicographic order
- union maps
  - single object
  - strict lexicographic order
  - schedule transformations can be composed
- schedule trees
  - single object
  - relaxed lexicographic order

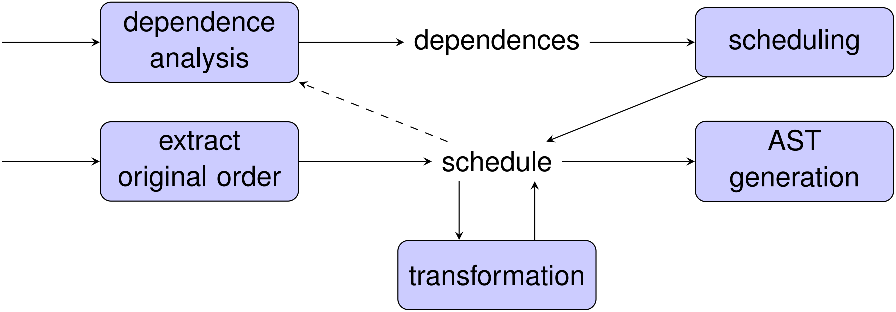
Outline

1. Introduction
   - Example
   - Single Statement
   - Multiple Statements
   - Schedule Trees
2. Advantages
   - Useful in several contexts
   - More natural
   - More convenient
   - More expressive
   - Extensible
3. Conclusion

Schedule Uses

- Representing the original execution order
  - Input to dependence analysis (in `isl`)
  - Basis for manual/incremental transformations
- Scheduling
  - Construction based on dependences
  - Schedule modifications
- AST generation
  - Generate AST from schedule

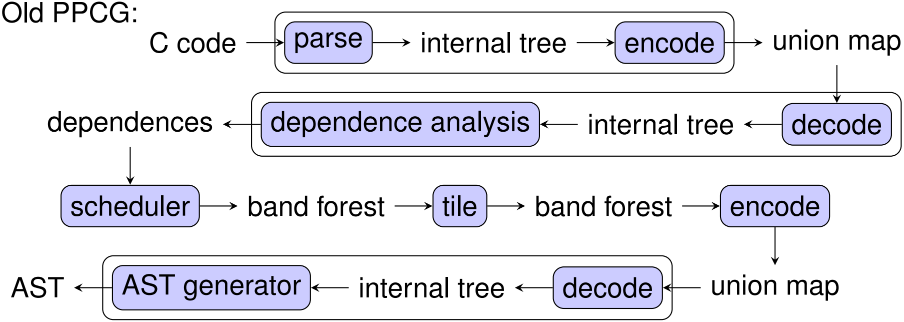
Schedule Uses



- Representing the original execution order
    - Input to dependence analysis (in `isl`)
    - Basis for manual/incremental transformations
- Scheduling
    - Construction based on dependences
    - Schedule modifications
- AST generation
    - Generate AST from schedule

Schedule Trees Everywhere

Old PPCG:



C code → parse → internal tree → encode → union map

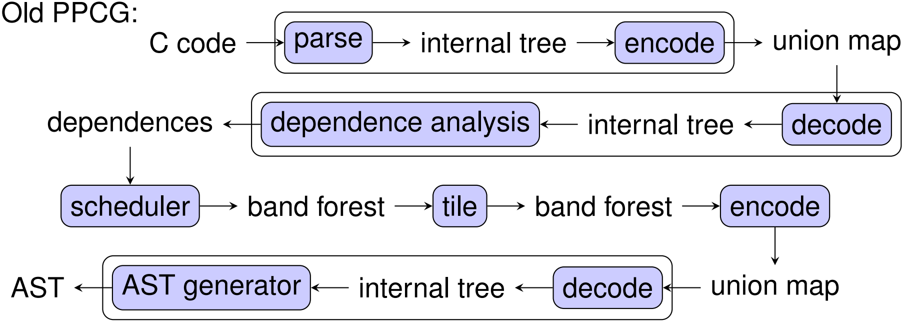
union map → decode → internal tree → dependence analysis → dependences

dependences → scheduler → band forest → tile → band forest → encode → union map

union map → decode → internal tree → AST generator → AST

Schedule Trees Everywhere

Old PPCG:



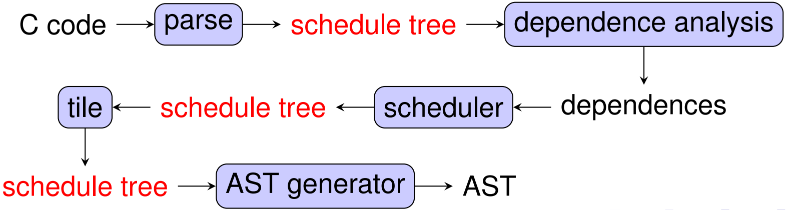
C code → parse → internal tree → encode → union map

union map → decode → internal tree → dependence analysis → dependences

dependences → scheduler → band forest → tile → band forest → encode → union map

union map → decode → internal tree → AST generator → AST

New PPCG:



C code → parse → schedule tree → dependence analysis → dependences

dependences → scheduler → schedule tree → tile → schedule tree

schedule tree → AST generator → AST

Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

$$\{\, \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N; \mathtt{U}[] \,\}$$

- Dependences

$$\{\, \mathtt{S}[i] \to \mathtt{T}[N-i] : 0 \le i \le N \,\}$$

Schedule Construction Example

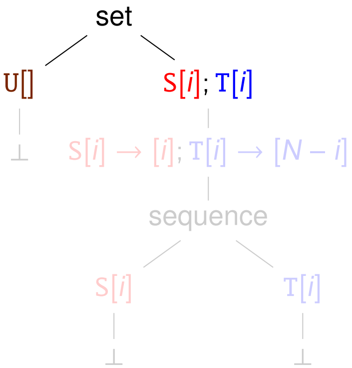
```
  for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
  for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

$$\{\, \mathtt{S}[i] : 0 \leq i \leq N; \mathtt{T}[i] : 0 \leq i \leq N; \mathtt{U}[] \,\}$$

- Dependences

$$\{\, \mathtt{S}[i] \rightarrow \mathtt{T}[N-i] : 0 \leq i \leq N \,\}$$



set

$\mathtt{U}[]$

$\mathtt{S}[i]; \mathtt{T}[i]$

$\perp$

$\mathtt{S}[i] \rightarrow [i]; \mathtt{T}[i] \rightarrow [N-i]$

sequence

$\mathtt{S}[i]$

$\mathtt{T}[i]$

$\perp$

$\perp$

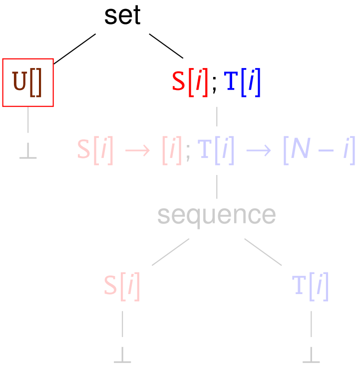
Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

$$\{ \mathtt{S}[i] : 0 \leq i \leq N; \mathtt{T}[i] : 0 \leq i \leq N; \mathtt{U}[] \}$$

- Dependences



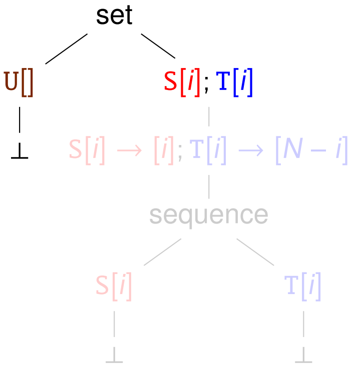
Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

$$\{\, S[i] : 0 \le i \le N; T[i] : 0 \le i \le N; U[\,] \,\}$$

- Dependences



set

U[]

⊥

S[i]; T[i]

S[i] → [i]; T[i] → [$N-i$]

sequence

S[i]

⊥

T[i]

⊥

Schedule Construction Example

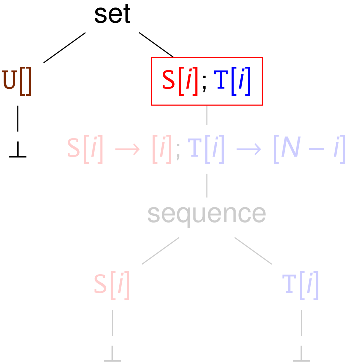
```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

  $$\{ \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N; \mathtt{U}[\,] \}$$

- Dependences

  $$\{ \mathtt{S}[i] \to \mathtt{T}[N - i] : 0 \le i \le N \}$$



set

$\mathtt{U}[\,]$

$\perp$

$\boxed{\mathtt{S}[i]; \mathtt{T}[i]}$

$\mathtt{S}[i] \to [i]; \mathtt{T}[i] \to [N - i]$

sequence

$\mathtt{S}[i]$

$\mathtt{T}[i]$

$\perp$

$\perp$

Schedule Construction Example

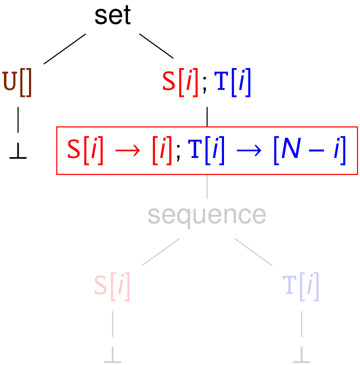
```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

$$\{\, S[i] : 0 \le i \le N;\, T[i] : 0 \le i \le N;\, U[\,]\,\}$$

- Dependences

$$\{\, S[i] \to T[N-i] : 0 \le i \le N \,\}$$



Schedule Construction Example

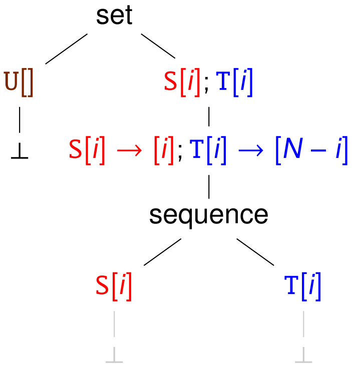
```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

  $$\{ \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N; \mathtt{U}[] \}$$

- Dependences

  $$\{ \mathtt{S}[i] \to \mathtt{T}[N-i] : 0 \le i \le N \}$$



set

$\mathtt{U}[]$

$\perp$

$\mathtt{S}[i]; \mathtt{T}[i]$

$\mathtt{S}[i] \to [i]; \mathtt{T}[i] \to [N-i]$

sequence

$\mathtt{S}[i]$

$\perp$

$\mathtt{T}[i]$

$\perp$

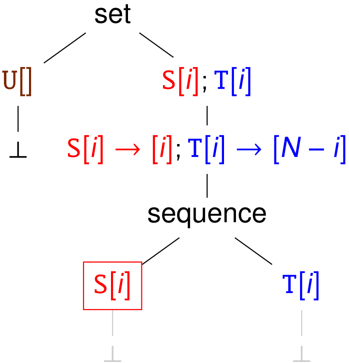
Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:      a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:      b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

$$\{\, S[i] : 0 \le i \le N; T[i] : 0 \le i \le N; U[\,] \,\}$$

- Dependences



set

U[]

$S[i]; T[i]$

$\perp$

$S[i] \to [i]; T[i] \to [N - i]$

sequence

$\boxed{S[i]}$

$T[i]$

$\perp$

$\perp$

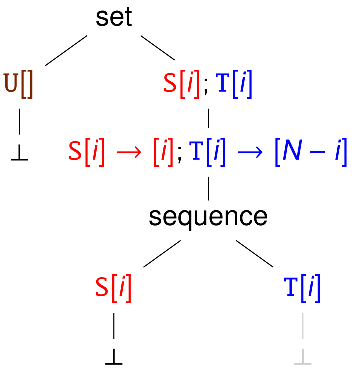
Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

  $\{ \mathrm{S}[i] : 0 \leq i \leq N; \mathrm{T}[i] : 0 \leq i \leq N; \mathrm{U}[] \}$

- Dependences



set

$\mathrm{U}[]$

$\perp$

$\mathrm{S}[i]; \mathrm{T}[i]$

$\mathrm{S}[i] \rightarrow [i]; \mathrm{T}[i] \rightarrow [N-i]$

sequence

$\mathrm{S}[i]$

$\perp$

$\mathrm{T}[i]$

$\perp$

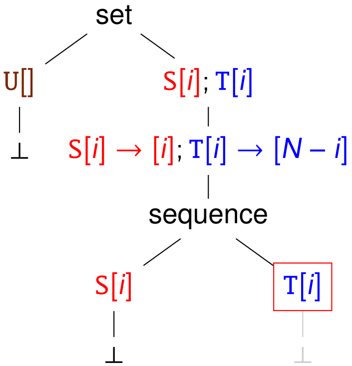
Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

$$\{\, \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N; \mathtt{U}[] \,\}$$

- Dependences



set

$\mathtt{U}[]$

$\perp$

$\mathtt{S}[i]; \mathtt{T}[i]$

$\mathtt{S}[i] \to [i]; \mathtt{T}[i] \to [N - i]$

sequence

$\mathtt{S}[i]$

$\perp$

$\boxed{\mathtt{T}[i]}$

$\perp$

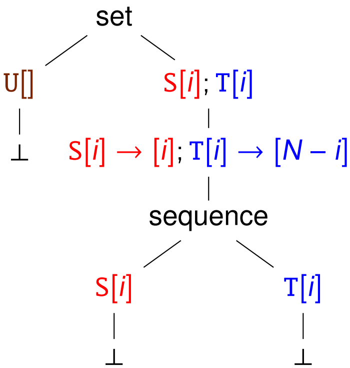
Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:       a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:       b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

  $\{\, S[i] : 0 \le i \le N; T[i] : 0 \le i \le N; U[] \,\}$

- Dependences



set

$U[]$

$S[i]; T[i]$

$\perp$

$S[i] \to [i]; T[i] \to [N - i]$

sequence

$S[i]$

$T[i]$

$\perp$

$\perp$

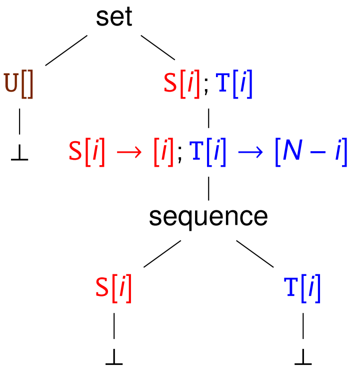
Schedule Construction Example

```
   for (i = 0; i <= N; ++i)
S:     a[i] = g(i);
   for (i = 0; i <= N; ++i)
T:     b[i] = f(a[N-i]);
U:c = 0;
```

- Iteration domain

  $$\{\, \mathtt{S}[i] : 0 \le i \le N; \mathtt{T}[i] : 0 \le i \le N; \mathtt{U}[\,] \,\}$$

- Dependences



⇒ natural representation of constructed schedule

Local Transformations

Typical scenario:

1. Construct tilable bands (e.g., using Pluto algorithm)
2. Individually tile (some) tilable bands
    - Given a band $D(\mathbf{i}) \to \mathbf{f}(\mathbf{i})$, insert a band $D(\mathbf{i}) \to \lfloor \mathbf{f}(\mathbf{i})/\mathbf{S} \rfloor$
    - First iterate over blocks of size **S** and then iterate within each block

Local Transformations

Typical scenario:

1. Construct tilable bands (e.g., using Pluto algorithm)
2. Individually tile (some) tilable bands
    - Given a band $D(\mathbf{i}) \to \mathbf{f}(\mathbf{i})$, insert a band $D(\mathbf{i}) \to \lfloor \mathbf{f}(\mathbf{i})/\mathbf{S} \rfloor$
    - First iterate over blocks of size **S** and then iterate within each block

Tiled individually:

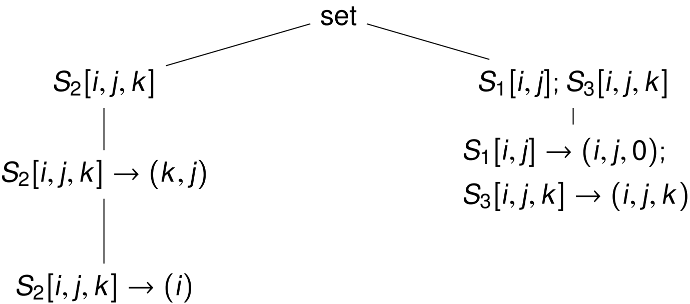
- bands of different dimensionality
- different tile sizes **S** per band

Local Transformations

Typical scenario:

1. Construct tilable bands (e.g., using Pluto algorithm)
2. Individually tile (some) tilable bands
   - Given a band $D(\mathbf{i}) \to \mathbf{f}(\mathbf{i})$, insert a band $D(\mathbf{i}) \to \lfloor \mathbf{f}(\mathbf{i})/\mathbf{S} \rfloor$
   - First iterate over blocks of size $\mathbf{S}$ and then iterate within each block

   Tiled individually:
   - bands of different dimensionality
   - different tile sizes $\mathbf{S}$ per band



set

$S_2[i,j,k]$

$S_2[i,j,k] \to (k,j)$

$S_2[i,j,k] \to (i)$

$S_1[i,j]; S_3[i,j,k]$

$S_1[i,j] \to (i,j,0);$

$S_3[i,j,k] \to (i,j,k)$

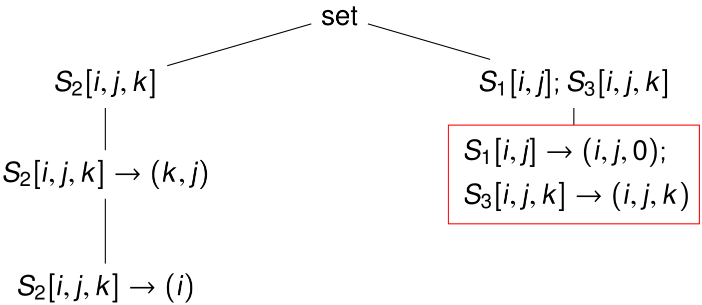
Local Transformations

Typical scenario:

1. Construct tilable bands (e.g., using Pluto algorithm)
2. Individually tile (some) tilable bands
    - Given a band $D(\mathbf{i}) \to \mathbf{f}(\mathbf{i})$, insert a band $D(\mathbf{i}) \to \lfloor \mathbf{f}(\mathbf{i})/\mathbf{S} \rfloor$
    - First iterate over blocks of size $\mathbf{S}$ and then iterate within each block

Tiled individually:

- bands of different dimensionality
- different tile sizes $\mathbf{S}$ per band



set

$S_2[i,j,k]$

$S_2[i,j,k] \to (k,j)$

$S_2[i,j,k] \to (i)$

$S_1[i,j]; S_3[i,j,k]$

$S_1[i,j] \to (i,j,0);$

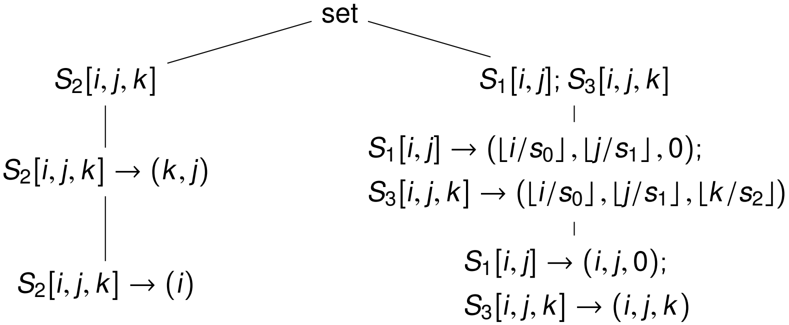
$S_3[i,j,k] \to (i,j,k)$

Local Transformations

Typical scenario:

1. Construct tilable bands (e.g., using Pluto algorithm)
2. Individually tile (some) tilable bands
   - Given a band $D(\mathbf{i}) \to \mathbf{f}(\mathbf{i})$, insert a band $D(\mathbf{i}) \to \lfloor \mathbf{f}(\mathbf{i})/\mathbf{S} \rfloor$
   - First iterate over blocks of size **S** and then iterate within each block

   Tiled individually:
   - bands of different dimensionality
   - different tile sizes **S** per band



set

$S_2[i,j,k]$

$S_2[i,j,k] \to (k,j)$

$S_2[i,j,k] \to (i)$

$S_1[i,j]; S_3[i,j,k]$

$S_1[i,j] \to (\lfloor i/s_0 \rfloor, \lfloor j/s_1 \rfloor, 0);$

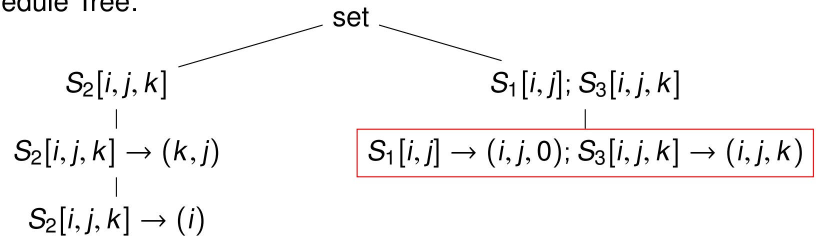
$S_3[i,j,k] \to (\lfloor i/s_0 \rfloor, \lfloor j/s_1 \rfloor, \lfloor k/s_2 \rfloor)$

$S_1[i,j] \to (i,j,0);$

$S_3[i,j,k] \to (i,j,k)$

Local Transformations

Schedule Tree:



set

$S_2[i, j, k]$

$S_2[i, j, k] \rightarrow (k, j)$

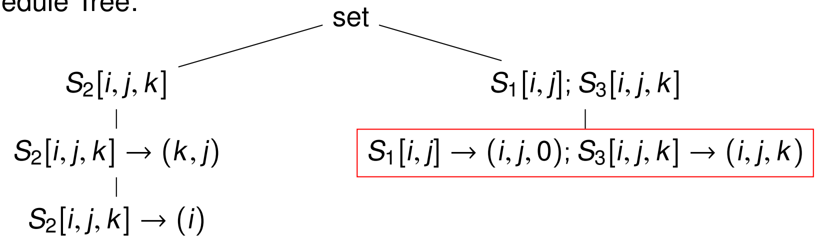
$S_2[i, j, k] \rightarrow (i)$

$S_1[i, j]; S_3[i, j, k]$

$\boxed{S_1[i, j] \rightarrow (i, j, 0); S_3[i, j, k] \rightarrow (i, j, k)}$

Local Transformations

Schedule Tree:



set

$S_2[i,j,k]$

$S_2[i,j,k] \to (k,j)$

$S_2[i,j,k] \to (i)$

$S_1[i,j]; S_3[i,j,k]$

$\boxed{S_1[i,j] \to (i,j,0); S_3[i,j,k] \to (i,j,k)}$

Kelly's abstraction:

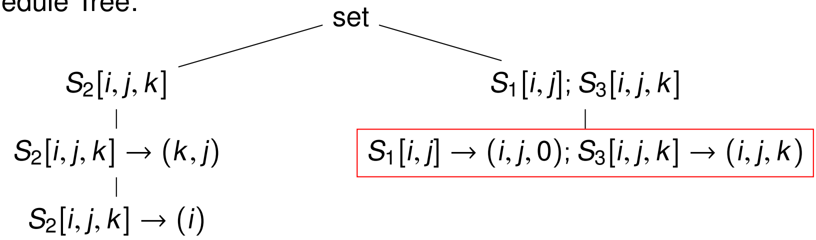
$$\begin{aligned} T_1 &: \{\, [i,j] \to [1,i,j,0] \,\} \\ T_2 &: \{\, [i,j,k] \to [0,k,j,i] \,\} \\ T_3 &: \{\, [i,j,k] \to [1,i,j,k] \,\} \end{aligned}$$

How to identify node that needs to be tiled?

- interval of dimensions
- list of statements or values for set/sequence encodings

Local Transformations

Schedule Tree:



set

$S_2[i,j,k]$

$S_2[i,j,k] \to (k,j)$

$S_2[i,j,k] \to (i)$

$S_1[i,j]; S_3[i,j,k]$

$\boxed{S_1[i,j] \to (i,j,0); S_3[i,j,k] \to (i,j,k)}$

Kelly's abstraction:

$$\begin{aligned} T_1 &: \{\, [i,j] \to [1,i,j,0] \,\} \\ T_2 &: \{\, [i,j,k] \to [0,k,j,i] \,\} \\ T_3 &: \{\, [i,j,k] \to [1,i,j,k] \,\} \end{aligned}$$

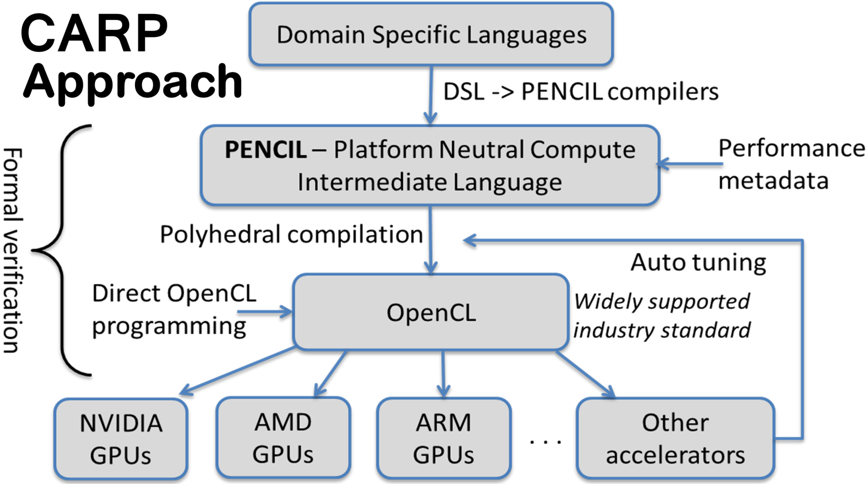
How to identify node that needs to be tiled?

- interval of dimensions
- list of statements or values for set/sequence encodings

Union map representation additionally requires alignment of single schedule space

CARP Project

Design tools and techniques to aid
Correct and Efficient Accelerator Programming



Advanced Use: CUDA/OpenCL Code Generation

- Schedule tree logically split into two parts
  - Outer part mapped to host code
  - Subtrees mapped to device code
- Device part has additional symbolic constants
  - ⇒ block and thread identifiers
  - ⇒ internal context nodes
- Each thread executes only part of iteration domain
  - ⇒ selected using filter nodes

Advanced Use: CUDA/OpenCL Code Generation

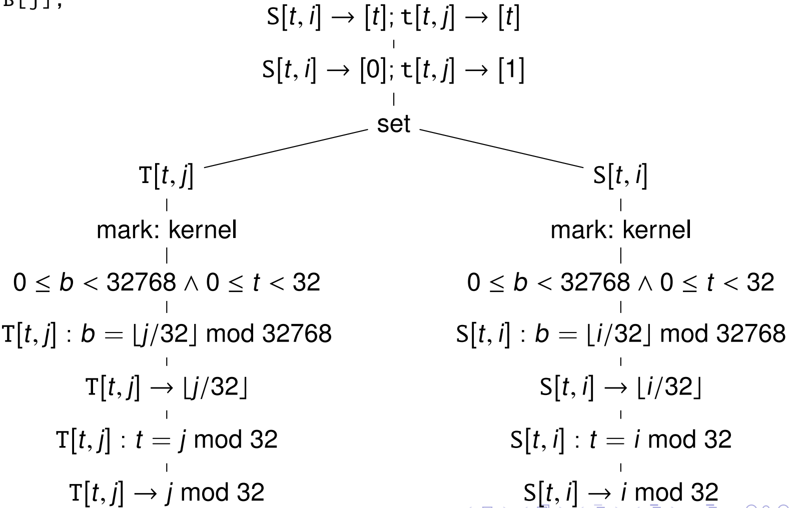
- Schedule tree logically split into two parts
    - Outer part mapped to host code
    - Subtrees mapped to device code
- Device part has additional symbolic constants
    - ⇒ block and thread identifiers
    - ⇒ internal context nodes
- Each thread executes only part of iteration domain
    - ⇒ selected using filter nodes

Old PPCG used nested AST generation

⇒ difficult to understand and debug

Advanced Use: CUDA/OpenCL Code Generation

```
for (t = 0; t < T; t++) {
  for (i = 1; i < N - 1; i++)
    B[i] = 0.33333 * (A[i-1] + A[i] + A[i + 1]);
  for (j = 1; j < N - 1; j++)
    A[j] = B[j];
}
```



$\mathtt{S}[t,i] \to [t]; \mathtt{T}[t,j] \to [t]$

$\mathtt{S}[t,i] \to [0]; \mathtt{T}[t,j] \to [1]$

set

$\mathtt{T}[t,j]$

mark: kernel

$0 \le b < 32768 \wedge 0 \le t < 32$

$\mathtt{T}[t,j] : b = \lfloor j/32 \rfloor \bmod 32768$

$\mathtt{T}[t,j] \to \lfloor j/32 \rfloor$

$\mathtt{T}[t,j] : t = j \bmod 32$

$\mathtt{T}[t,j] \to j \bmod 32$

$\mathtt{S}[t,i]$

mark: kernel

$0 \le b < 32768 \wedge 0 \le t < 32$

$\mathtt{S}[t,i] : b = \lfloor i/32 \rfloor \bmod 32768$

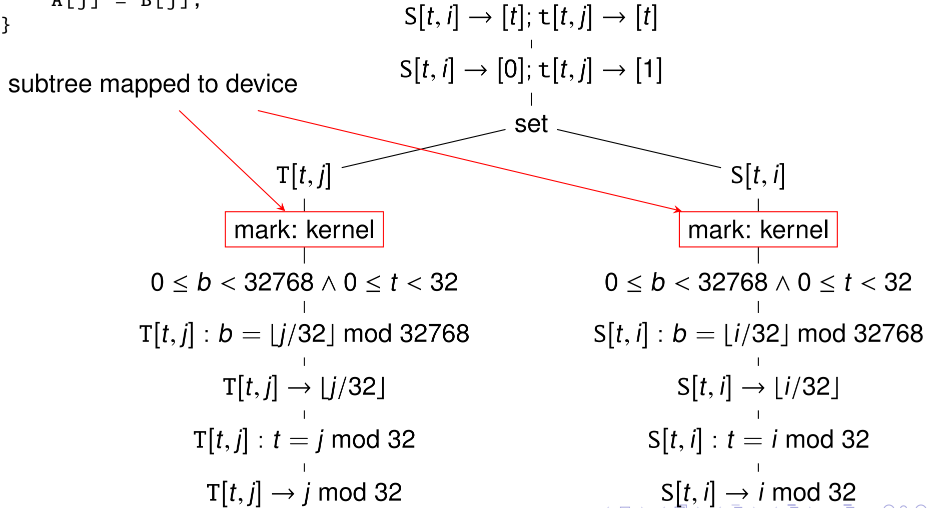
$\mathtt{S}[t,i] \to \lfloor i/32 \rfloor$

$\mathtt{S}[t,i] : t = i \bmod 32$

$\mathtt{S}[t,i] \to i \bmod 32$

Advanced Use: CUDA/OpenCL Code Generation

```
for (t = 0; t < T; t++) {
  for (i = 1; i < N - 1; i++)
    B[i] = 0.33333 * (A[i-1] + A[i] + A[i + 1]);
  for (j = 1; j < N - 1; j++)
    A[j] = B[j];
}
```



$\mathtt{S}[t,i] \to [t]; \mathtt{T}[t,j] \to [t]$

$\mathtt{S}[t,i] \to [0]; \mathtt{T}[t,j] \to [1]$

set

$\mathtt{T}[t,j]$

mark: kernel

$0 \le b < 32768 \wedge 0 \le t < 32$

$\mathtt{T}[t,j] : b = \lfloor j/32 \rfloor \bmod 32768$

$\mathtt{T}[t,j] \to \lfloor j/32 \rfloor$

$\mathtt{T}[t,j] : t = j \bmod 32$

$\mathtt{T}[t,j] \to j \bmod 32$

$\mathtt{S}[t,i]$

mark: kernel

$0 \le b < 32768 \wedge 0 \le t < 32$

$\mathtt{S}[t,i] : b = \lfloor i/32 \rfloor \bmod 32768$

$\mathtt{S}[t,i] \to \lfloor i/32 \rfloor$

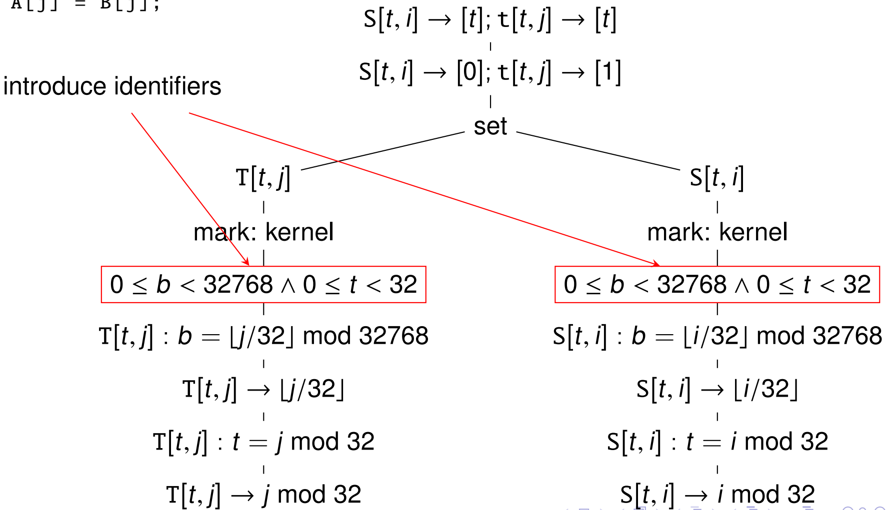
$\mathtt{S}[t,i] : t = i \bmod 32$

$\mathtt{S}[t,i] \to i \bmod 32$

subtree mapped to device

Advanced Use: CUDA/OpenCL Code Generation

```
for (t = 0; t < T; t++) {
  for (i = 1; i < N - 1; i++)
    B[i] = 0.33333 * (A[i-1] + A[i] + A[i + 1]);
  for (j = 1; j < N - 1; j++)
    A[j] = B[j];
}
```



$\texttt{S}[t,i] \to [t]; \texttt{T}[t,j] \to [t]$

$\texttt{S}[t,i] \to [0]; \texttt{T}[t,j] \to [1]$

introduce identifiers

set

$\texttt{T}[t,j]$

mark: kernel

$\boxed{0 \le b < 32768 \wedge 0 \le t < 32}$

$\texttt{T}[t,j] : b = \lfloor j/32 \rfloor \bmod 32768$

$\texttt{T}[t,j] \to \lfloor j/32 \rfloor$

$\texttt{T}[t,j] : t = j \bmod 32$

$\texttt{T}[t,j] \to j \bmod 32$

$\texttt{S}[t,i]$

mark: kernel

$\boxed{0 \le b < 32768 \wedge 0 \le t < 32}$

$\texttt{S}[t,i] : b = \lfloor i/32 \rfloor \bmod 32768$

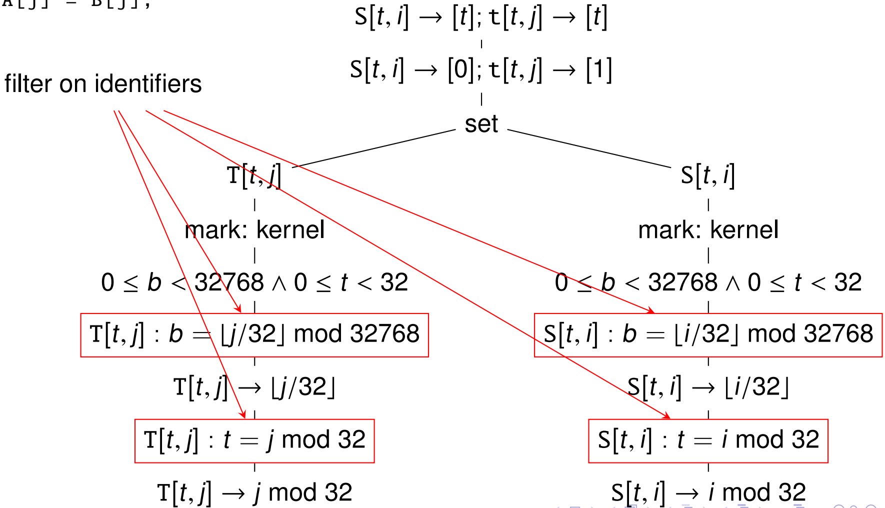
$\texttt{S}[t,i] \to \lfloor i/32 \rfloor$

$\texttt{S}[t,i] : t = i \bmod 32$

$\texttt{S}[t,i] \to i \bmod 32$

Advanced Use: CUDA/OpenCL Code Generation

```
for (t = 0; t < T; t++) {
  for (i = 1; i < N - 1; i++)
    B[i] = 0.33333 * (A[i-1] + A[i] + A[i + 1]);
  for (j = 1; j < N - 1; j++)
    A[j] = B[j];
}
```



filter on identifiers

$\mathtt{S}[t,i] \to [t]; \mathtt{t}[t,j] \to [t]$

$\mathtt{S}[t,i] \to [0]; \mathtt{t}[t,j] \to [1]$

set

$\mathtt{T}[t,j]$

mark: kernel

$0 \le b < 32768 \wedge 0 \le t < 32$

$\boxed{\mathtt{T}[t,j] : b = \lfloor j/32 \rfloor \bmod 32768}$

$\mathtt{T}[t,j] \to \lfloor j/32 \rfloor$

$\boxed{\mathtt{T}[t,j] : t = j \bmod 32}$

$\mathtt{T}[t,j] \to j \bmod 32$

$\mathtt{S}[t,i]$

mark: kernel

$0 \le b < 32768 \wedge 0 \le t < 32$

$\boxed{\mathtt{S}[t,i] : b = \lfloor i/32 \rfloor \bmod 32768}$

$\mathtt{S}[t,i] \to \lfloor i/32 \rfloor$

$\boxed{\mathtt{S}[t,i] : t = i \bmod 32}$

$\mathtt{S}[t,i] \to i \bmod 32$

Extension

In final stages of scheduling, additional statements may need to be added

- Copy code
- Synchronization
- ...

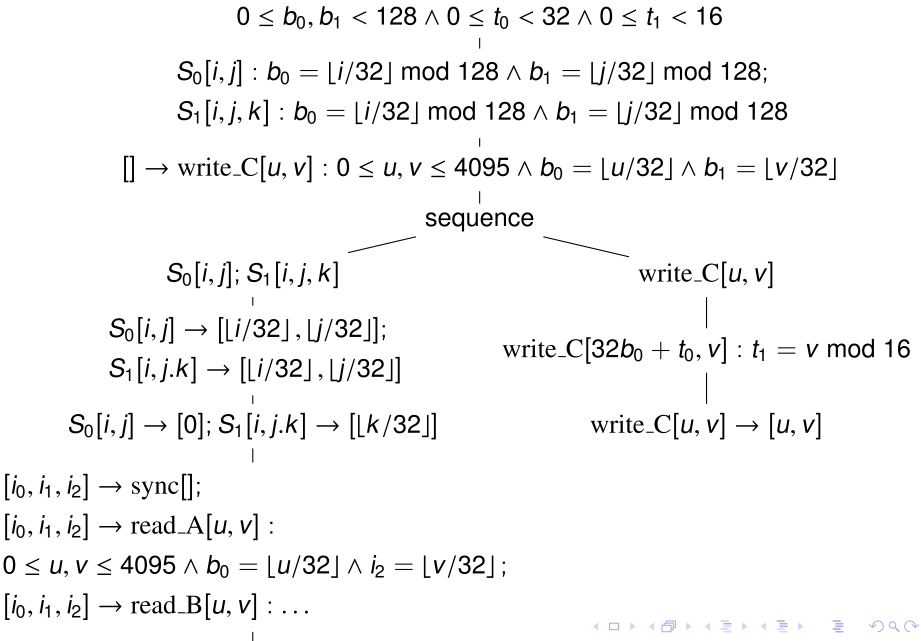
These additional statements depend on ancestors

- the statements should only be executed in a given part of the schedule tree
- iteration domains depend on outer schedule (e.g., data to be copied)

⇒ new “extension” node type

⇒ maps outer schedule dimensions to extra iteration domain

Extension



$$0 \leq b_0, b_1 < 128 \wedge 0 \leq t_0 < 32 \wedge 0 \leq t_1 < 16$$

$$S_0[i,j] : b_0 = \lfloor i/32 \rfloor \bmod 128 \wedge b_1 = \lfloor j/32 \rfloor \bmod 128;$$
$$S_1[i,j,k] : b_0 = \lfloor i/32 \rfloor \bmod 128 \wedge b_1 = \lfloor j/32 \rfloor \bmod 128$$

$$[] \rightarrow \text{write\_C}[u,v] : 0 \leq u, v \leq 4095 \wedge b_0 = \lfloor u/32 \rfloor \wedge b_1 = \lfloor v/32 \rfloor$$

sequence

$$S_0[i,j]; S_1[i,j,k]$$

$$S_0[i,j] \rightarrow [\lfloor i/32 \rfloor, \lfloor j/32 \rfloor];$$
$$S_1[i,j.k] \rightarrow [\lfloor i/32 \rfloor, \lfloor j/32 \rfloor]$$

$$S_0[i,j] \rightarrow [0]; S_1[i,j.k] \rightarrow [\lfloor k/32 \rfloor]$$

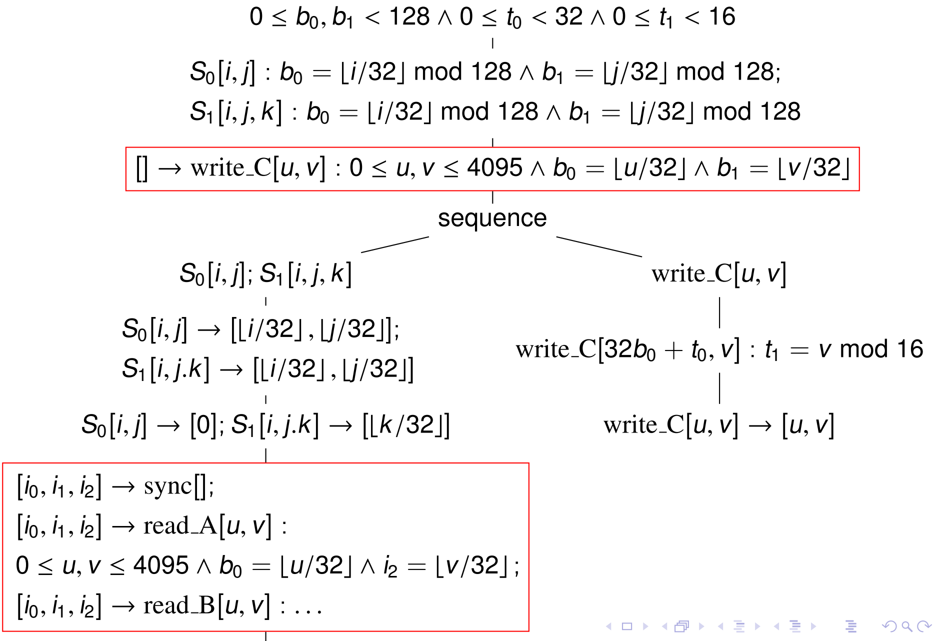
$$[i_0, i_1, i_2] \rightarrow \text{sync}[];$$
$$[i_0, i_1, i_2] \rightarrow \text{read\_A}[u,v] :$$
$$0 \leq u, v \leq 4095 \wedge b_0 = \lfloor u/32 \rfloor \wedge i_2 = \lfloor v/32 \rfloor;$$
$$[i_0, i_1, i_2] \rightarrow \text{read\_B}[u,v] : \ldots$$

$$\text{write\_C}[u,v]$$

$$\text{write\_C}[32b_0 + t_0, v] : t_1 = v \bmod 16$$

$$\text{write\_C}[u,v] \rightarrow [u,v]$$

Extension



$$0 \leq b_0, b_1 < 128 \wedge 0 \leq t_0 < 32 \wedge 0 \leq t_1 < 16$$

$$S_0[i,j] : b_0 = \lfloor i/32 \rfloor \bmod 128 \wedge b_1 = \lfloor j/32 \rfloor \bmod 128;$$
$$S_1[i,j,k] : b_0 = \lfloor i/32 \rfloor \bmod 128 \wedge b_1 = \lfloor j/32 \rfloor \bmod 128$$

$$\boxed{[] \to \mathrm{write\_C}[u,v] : 0 \leq u, v \leq 4095 \wedge b_0 = \lfloor u/32 \rfloor \wedge b_1 = \lfloor v/32 \rfloor}$$

sequence

$$S_0[i,j]; S_1[i,j,k]$$

$$S_0[i,j] \to [\lfloor i/32 \rfloor, \lfloor j/32 \rfloor];$$
$$S_1[i,j.k] \to [\lfloor i/32 \rfloor, \lfloor j/32 \rfloor]$$

$$S_0[i,j] \to [0]; S_1[i,j.k] \to [\lfloor k/32 \rfloor]$$

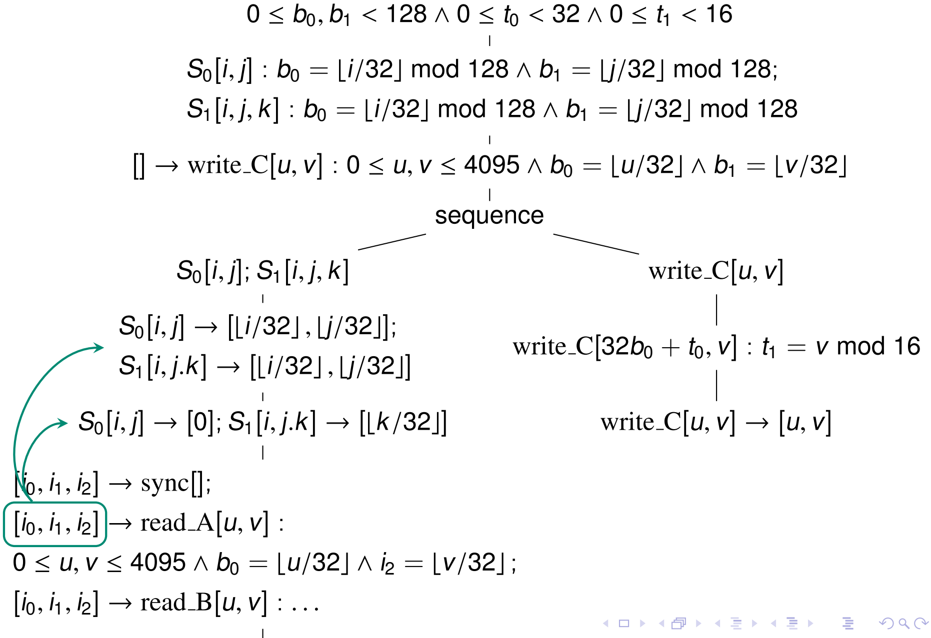
$$\boxed{\begin{array}{l} [i_0, i_1, i_2] \to \mathrm{sync}[]; \\ [i_0, i_1, i_2] \to \mathrm{read\_A}[u,v] : \\ 0 \leq u, v \leq 4095 \wedge b_0 = \lfloor u/32 \rfloor \wedge i_2 = \lfloor v/32 \rfloor; \\ [i_0, i_1, i_2] \to \mathrm{read\_B}[u,v] : \ldots \end{array}}$$

$$\mathrm{write\_C}[u,v]$$

$$\mathrm{write\_C}[32b_0 + t_0, v] : t_1 = v \bmod 16$$

$$\mathrm{write\_C}[u,v] \to [u,v]$$

Extension



$$0 \le b_0, b_1 < 128 \wedge 0 \le t_0 < 32 \wedge 0 \le t_1 < 16$$

$$S_0[i,j] : b_0 = \lfloor i/32 \rfloor \bmod 128 \wedge b_1 = \lfloor j/32 \rfloor \bmod 128;$$

$$S_1[i,j,k] : b_0 = \lfloor i/32 \rfloor \bmod 128 \wedge b_1 = \lfloor j/32 \rfloor \bmod 128$$

$$[] \to \mathrm{write\_C}[u,v] : 0 \le u, v \le 4095 \wedge b_0 = \lfloor u/32 \rfloor \wedge b_1 = \lfloor v/32 \rfloor$$

sequence

$$S_0[i,j]; S_1[i,j,k]$$

$$S_0[i,j] \to [\lfloor i/32 \rfloor, \lfloor j/32 \rfloor];$$

$$S_1[i,j.k] \to [\lfloor i/32 \rfloor, \lfloor j/32 \rfloor]$$

$$S_0[i,j] \to [0]; S_1[i,j.k] \to [\lfloor k/32 \rfloor]$$

$$[i_0, i_1, i_2] \to \mathrm{sync}[];$$

$$\boxed{[i_0, i_1, i_2]} \to \mathrm{read\_A}[u,v] :$$

$$0 \le u, v \le 4095 \wedge b_0 = \lfloor u/32 \rfloor \wedge i_2 = \lfloor v/32 \rfloor;$$

$$[i_0, i_1, i_2] \to \mathrm{read\_B}[u,v] : \ldots$$

$$\mathrm{write\_C}[u,v]$$

$$\mathrm{write\_C}[32b_0 + t_0, v] : t_1 = v \bmod 16$$

$$\mathrm{write\_C}[u,v] \to [u,v]$$

Outline

1. Introduction
   - Example
   - Single Statement
   - Multiple Statements
   - Schedule Trees
2. Advantages
   - Useful in several contexts
   - More natural
   - More convenient
   - More expressive
   - Extensible
3. Conclusion

Conclusion

Conclusion:

Exploit the tree nature of a schedule rather than encoding it in a flat representation

Schedule trees are

- useful in several contexts
- more natural
- more convenient
- more expressive
- extensible

Conclusion

Conclusion:

Exploit the tree nature of a schedule rather than encoding it in a flat representation

Schedule trees are

- useful in several contexts
- more natural
- more convenient
- more expressive
- extensible

Future work

- apply separation on schedule tree
- additional node types
    - parametric tiling
    - clustering
    - …