Live-Range Reordering

Sven Verdoolaege[1] Albert Cohen[2]

[1]Polly Labs and KU Leuven

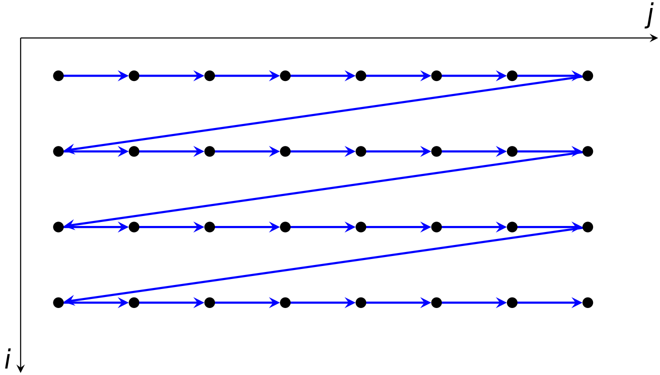
[2]INRIA and École Normale Supérieure

January 19, 2016

Outline

Outline

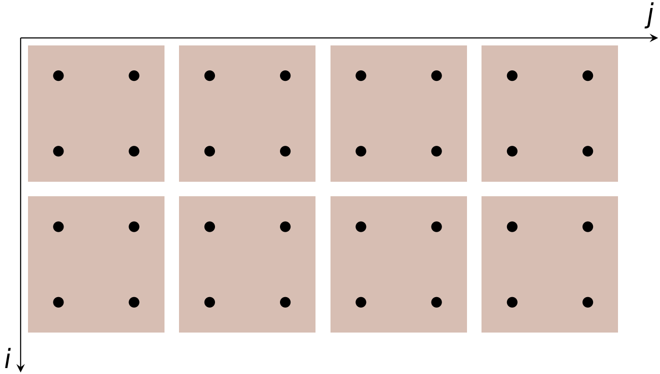
Tiling Intuition



Assume reuse along rows and columns

→: execution order

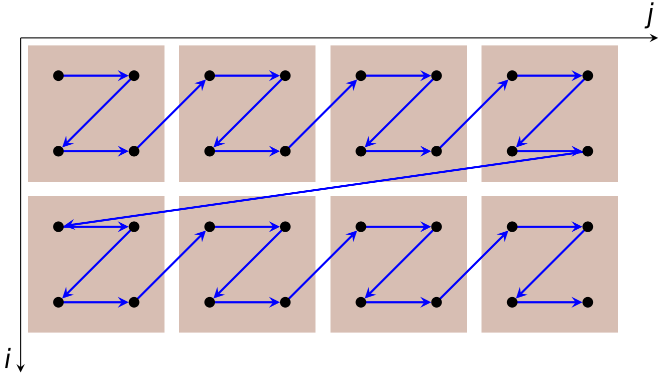
Tiling Intuition



Assume reuse along rows and columns

⟶: execution order

Tiling Intuition



Assume reuse along rows and columns

$\longrightarrow$: execution order

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(symm.c from PolyBench/C 4.1)

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(symm.c from PolyBench/C 4.1)

After tiling:

```
for (int c0 = 0; c0 < m; c0 += 32)
  for (int c1 = 0; c1 < n; c1 += 32)
    for (int c2 = 0; c2 <= min(31, m - c0 - 1); c2 += 1)
      for (int c3 = 0; c3 <= min(31, n - c1 - 1); c3 += 1) {
        temp2 = 0;
        for (int c4 = 0; c4 < c0 + c2; c4 += 1) {
          C[c4][c1 + c3] += ((alpha * B[c0 + c2][c1 + c3]) * A[c0 + c2][c4
          temp2 += (B[c4][c1 + c3] * A[c0 + c2][c4]);
        }
        C[c0 + c2][c1 + c3] = (((beta * C[c0 + c2][c1 + c3]) + ((alpha * E
      }
```

Schedule Constraints

Tiling is a form of restructuring loop transformation

$\Rightarrow$ changes execution order of statement instances

$\Rightarrow$ needs to preserve semantics

$\Rightarrow$ impose schedule constraints of the form

statement instance **a** needs to be executed before instance **b**

Schedule Constraints

Tiling is a form of restructuring loop transformation

$\Rightarrow$ changes execution order of statement instances

$\Rightarrow$ needs to preserve semantics

$\Rightarrow$ impose schedule constraints of the form

statement instance **a** needs to be executed before instance **b**

In particular, any statement instance writing a value should be executed before any statement instance reading that value

$\Rightarrow$ *flow dependences* aka *live ranges*

Schedule Constraints

Tiling is a form of restructuring loop transformation

- ⇒ changes execution order of statement instances
- ⇒ needs to preserve semantics
- ⇒ impose schedule constraints of the form

statement instance **a** needs to be executed before instance **b**

In particular, any statement instance writing a value should be executed before any statement instance reading that value

- ⇒ *flow dependences* aka *live ranges*

Moreover, no write from **before** or **after** the live-range should be moved **inside** the live-range

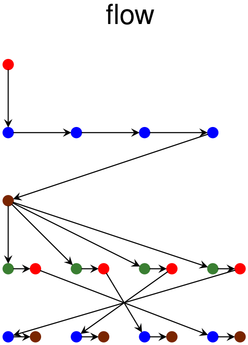
- ⇒ traditionally,
  - *output dependences* between two writes to same location
  - *anti-dependences* between reads and subsequent writes to same location

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```

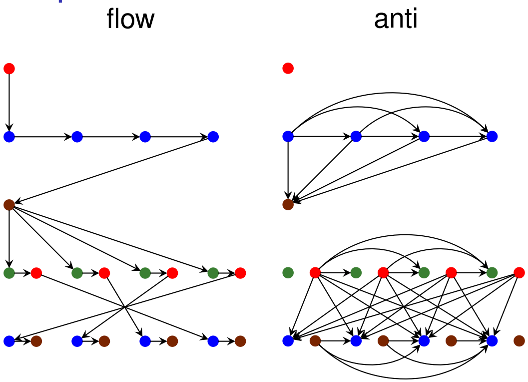
Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
    avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
    tmp = A[i] - avg;
    A[i] = tmp;
}
for (i=0; i<N; ++i) {
    tmp = A[N - 1 - i];
    B[i] = tmp;
}
```



Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(symm.c from PolyBench/C 4.1)

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(`symm.c` from PolyBench/C 4.1)

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(`symm.c` from PolyBench/C 4.1)

- ⇒ anti-dependence between every instance of statement reading `temp2` and every later instance writing to `temp2`
- ⇒ serialized execution order

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(`symm.c` from PolyBench/C 4.1)

$\Rightarrow$ anti-dependence between every instance of statement reading `temp2` and every later instance writing to `temp2`

$\Rightarrow$ serialized execution order

Such serializing anti-dependences are very common in practice

$\Rightarrow$ occur in nearly all experiments of Baghdadi, Beaugnon, et al. (2015)

$\Rightarrow$ no optimization possible without alternative to anti-dependences

Outline

1. Introduction
   - Example
   - Schedule Constraints

2. Live Range Reordering
   - Related Work
   - Scheduling
   - Relaxed Permutability Criterion
   - Conditional Validity Constraints

3. Conclusion

Alternatives to Anti-Dependences

- Conversion to single assignment through expansion (possibly followed by contraction)
  - \+ full scheduling freedom
  - (–) may increase memory requirements

Note: choice also has effect on scheduling time

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(symm.c from PolyBench/C 4.1)

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(symm.c from PolyBench/C 4.1)

After expansion:

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2[i][j][0] = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2[i][j][k+1] = temp[i][j][k] + B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2[i][j][i];
}
```

Alternatives to Anti-Dependences

- Conversion to single assignment through expansion (possibly followed by contraction)
  - + full scheduling freedom
  - (–) may increase memory requirements

Note: choice also has effect on scheduling time

Alternatives to Anti-Dependences

- Conversion to single assignment through expansion (possibly followed by contraction)
  - \+ full scheduling freedom
  - (–) may increase memory requirements
- Cluster live-range statements
  Note:
  - in general, clustering is partial scheduling
  - simple clusterings lead to coarse statements
  - \+ no increase in memory requirements
  - – significant loss of scheduling freedom

Note: choice also has effect on scheduling time

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(`symm.c` from PolyBench/C 4.1)

Tiling Example

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    temp2 = 0;
    for (k = 0; k < i; k++) {
       C[k][j] += alpha*B[i][j] * A[i][k];
       temp2 += B[k][j] * A[i][k];
    }
    C[i][j] = beta*C[i][j] + alpha*B[i][j]*A[i][i] + alpha*temp2;
}
```

(symm.c from PolyBench/C 4.1)

Alternatives to Anti-Dependences

- Conversion to single assignment through expansion (possibly followed by contraction)
  - $+$ full scheduling freedom
  - (–) may increase memory requirements
- Cluster live-range statements
  Note:
  - in general, clustering is partial scheduling
  - simple clusterings lead to coarse statements
  - $+$ no increase in memory requirements
  - – significant loss of scheduling freedom

Note: choice also has effect on scheduling time

Alternatives to Anti-Dependences

- Conversion to single assignment through expansion (possibly followed by contraction)
  - + full scheduling freedom
  - (–) may increase memory requirements
- Cluster live-range statements
  Note:
  - in general, clustering is partial scheduling
  - simple clusterings lead to coarse statements
  - + no increase in memory requirements
  - – significant loss of scheduling freedom
- Live-range reordering
  - + no increase in memory requirements
  - (–) limited loss of scheduling freedom

Note: choice also has effect on scheduling time

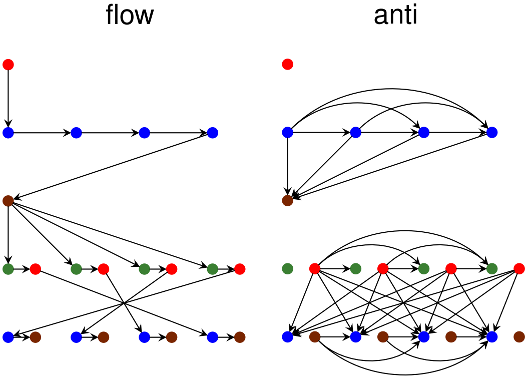
Live-Range Reordering

Basic idea:

allow live-ranges to be reordered with respect to each other as long as they do not overlap

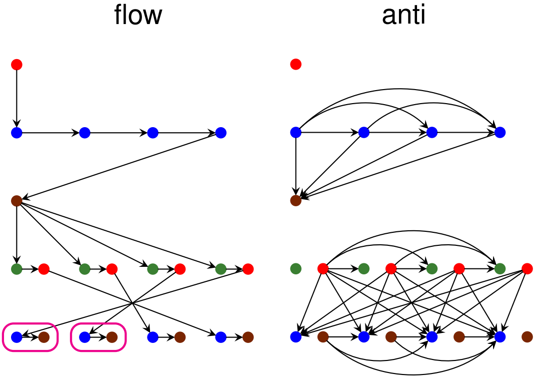
Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



Live-Range Reordering

Basic idea:

allow live-ranges to be reordered with respect to each other as long as they do not overlap

Live-Range Reordering

Basic idea:

allow live-ranges to be reordered with respect to each other as long as they do not overlap

- encode disjunction in scheduling problem (Baghdadi 2011)
- relaxed permutability criterion (Baghdadi, Cohen, et al. 2013)

  application by Baghdadi, Cohen, et al. (2013):
  - use standard scheduling algorithm
  - *reinterpret* results
- variable liberalization (Mehta 2014)
  - removes specific patterns of anti-dependences
- conditional validity constraints

Live-Range Reordering

Basic idea:

allow live-ranges to be reordered with respect to each other as long as they do not overlap

- encode disjunction in scheduling problem (Baghdadi 2011)
- relaxed permutability criterion (Baghdadi, Cohen, et al. 2013)

  application by Baghdadi, Cohen, et al. (2013):
  - use standard scheduling algorithm
  - *reinterpret* results
- variable liberalization (Mehta 2014)
  - removes specific patterns of anti-dependences
- conditional validity constraints

Scheduling

A schedule determines the *execution order* of statement instances and is expressed using a (recursive) combination of

- affine functions f
  $f(\mathbf{i}) < f(\mathbf{j}) \quad \Rightarrow \mathbf{i}$ executed before $\mathbf{j}$
- finite sequence $S_1, S_2, \ldots, S_n$
  $\mathbf{i} \in S_{k_1} \wedge \mathbf{j} \in S_{k_2} \wedge k_1 < k_2 \quad \Rightarrow \mathbf{i}$ executed before $\mathbf{j}$

Scheduling

A schedule determines the *execution order* of statement instances and is expressed using a (recursive) combination of

- affine functions f

  $f(\mathbf{i}) < f(\mathbf{j}) \quad \Rightarrow \mathbf{i}$ executed before $\mathbf{j}$

- finite sequence $S_1, S_2, \ldots, S_n$

  $\mathbf{i} \in S_{k_1} \wedge \mathbf{j} \in S_{k_2} \wedge k_1 < k_2 \quad \Rightarrow \mathbf{i}$ executed before $\mathbf{j}$

Scheduling determines schedule compatible with schedule constraints

statement instance $\mathbf{a}$ needs to be executed before instance $\mathbf{b}$

$\Rightarrow$ there is some node with

$f(\mathbf{a}) < f(\mathbf{b}) \quad$ or $\quad \mathbf{a} \in S_{k_1} \wedge \mathbf{b} \in S_{k_2} \wedge k_1 < k_2$

$\Rightarrow$ for all outer nodes

$f(\mathbf{a}) = f(\mathbf{b}) \quad$ or $\quad \exists k : \{\mathbf{a}, \mathbf{b}\} \subseteq S_k$

Scheduling

A schedule determines the *execution order* of statement instances and is expressed using a (recursive) combination of

- affine functions f a.k.a. band members
  $f(\mathbf{i}) < f(\mathbf{j}) \quad \Rightarrow$ **i** executed before **j**
- finite sequence $S_1, S_2, \ldots, S_n$
  $\mathbf{i} \in S_{k_1} \wedge \mathbf{j} \in S_{k_2} \wedge k_1 < k_2 \quad \Rightarrow$ **i** executed before **j**

Scheduling determines schedule compatible with schedule constraints

statement instance **a** needs to be executed before instance **b**

$\Rightarrow$ there is some node with

$$f(\mathbf{a}) < f(\mathbf{b}) \quad \text{or} \quad \mathbf{a} \in S_{k_1} \wedge \mathbf{b} \in S_{k_2} \wedge k_1 < k_2$$

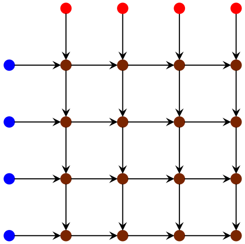
$\Rightarrow$ for all outer nodes

$$f(\mathbf{a}) = f(\mathbf{b}) \quad \text{or} \quad \exists k : \{\, \mathbf{a}, \mathbf{b} \,\} \subseteq S_k$$

Band: nested sequence of affine functions that can be freely reordered

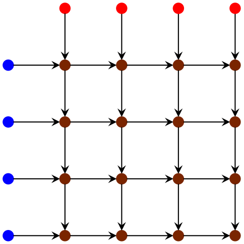
Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Schedule

$\mathtt{A}[i] \to i; \mathtt{B}[i] \to 0; \mathtt{C}[i,j] \to i$

$|$

$\{\,\mathtt{A}[i]\,\}, \{\,\mathtt{B}[i]\,\}, \{\,\mathtt{C}[i,j]\,\}$

Schedule constraints

$\mathtt{A}[i] \to \mathtt{C}[i,0]$

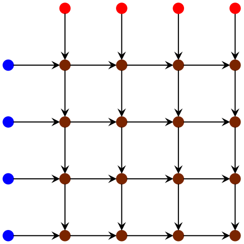
$\mathtt{B}[i] \to \mathtt{C}[0,i]$

$\mathtt{C}[i,j] \to \mathtt{C}[i+1,j]$

$\mathtt{C}[i,j] \to \mathtt{C}[i,j+1]$

Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Schedule

$\mathtt{A}[i] \to i; \mathtt{B}[i] \to 0; \mathtt{C}[i, j] \to i$

$\{\, \mathtt{A}[i] \,\}, \{\, \mathtt{B}[i] \,\}, \{\, \mathtt{C}[i, j] \,\}$

Schedule constraints

$\mathtt{A}[i] \to \mathtt{C}[i, 0]$

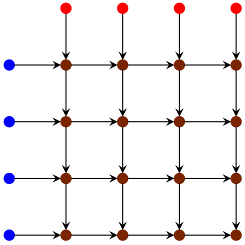
$\mathtt{B}[i] \to \mathtt{C}[0, i]$

$\mathtt{C}[i, j] \to \mathtt{C}[i + 1, j]$

$\mathtt{C}[i, j] \to \mathtt{C}[i, j + 1]$

Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Schedule

$A[i] \to i; B[i] \to 0; C[i,j] \to i$

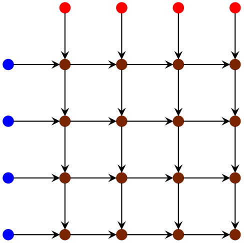
$\{ A[i] \}, \{ B[i] \}, \{ C[i,j] \}$

Schedule constraints

$A[i] \to C[i,0]$	$i \to i$
$B[i] \to C[0,i]$	$0 \to 0$
$C[i,j] \to C[i+1,j]$	$i \to i+1$
$C[i,j] \to C[i,j+1]$	$i \to i$

Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Schedule

$\mathtt{A}[i] \to i; \mathtt{B}[i] \to 0; \mathtt{C}[i, j] \to i$

$\mathtt{A}[i] \to 0; \mathtt{B}[i] \to i; \mathtt{C}[i, j] \to j$

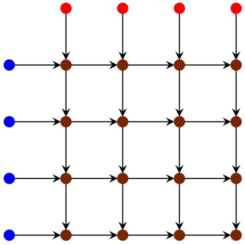
$\{ \mathtt{A}[i] \}, \{ \mathtt{B}[i] \}, \{ \mathtt{C}[i, j] \}$

Schedule constraints

$\mathtt{A}[i] \to \mathtt{C}[i, 0]$	$i \to i$
$\mathtt{B}[i] \to \mathtt{C}[0, i]$	$0 \to 0$
$\mathtt{C}[i, j] \to \mathtt{C}[i+1, j]$	$i \to i+1$
$\mathtt{C}[i, j] \to \mathtt{C}[i, j+1]$	$i \to i$

Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Schedule

$\mathtt{A}[i] \to i; \mathtt{B}[i] \to 0; \mathtt{C}[i,j] \to i$

$\mathtt{A}[i] \to 0; \mathtt{B}[i] \to i; \mathtt{C}[i,j] \to j$

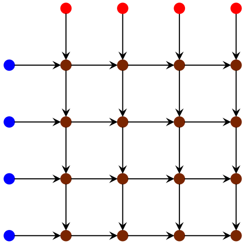
$\{\, \mathtt{A}[i] \,\}, \{\, \mathtt{B}[i] \,\}, \{\, \mathtt{C}[i,j] \,\}$

Schedule constraints

$\mathtt{A}[i] \to \mathtt{C}[i,0]$	$i \to i$	$0 \to 0$
$\mathtt{B}[i] \to \mathtt{C}[0,i]$	$0 \to 0$	$i \to i$
$\mathtt{C}[i,j] \to \mathtt{C}[i+1,j]$	$i \to i+1$	$j \to j$
$\mathtt{C}[i,j] \to \mathtt{C}[i,j+1]$	$i \to i$	$j \to j+1$

Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Schedule

$\mathtt{A}[i] \to i; \mathtt{B}[i] \to 0; \mathtt{C}[i,j] \to i$

$\mathtt{A}[i] \to 0; \mathtt{B}[i] \to i; \mathtt{C}[i,j] \to j$

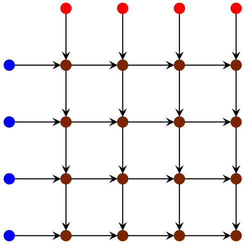
$\{ \mathtt{A}[i] \}, \{ \mathtt{B}[i] \}, \{ \mathtt{C}[i,j] \}$

Schedule constraints

$\mathtt{A}[i] \to \mathtt{C}[i,0]$	$i \to i$	$0 \to 0$
$\mathtt{B}[i] \to \mathtt{C}[0,i]$	$0 \to 0$	$i \to i$

Scheduling Example 1

```
for (i = 1; i < n; ++i)
A:M[i, 0] = f();
for (i = 1; i < n; ++i)
B:M[0, i] = g();
for (i = 1; i < n; ++i)
  for (j = 1; j < n; ++j)
C:  M[i][j] = h(M[i-1][j], M[i][j-1]);
```



Schedule

$\mathtt{A}[i] \to i; \mathtt{B}[i] \to 0; \mathtt{C}[i,j] \to i$

$\mathtt{A}[i] \to 0; \mathtt{B}[i] \to i; \mathtt{C}[i,j] \to j$

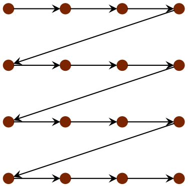
$\{\, \mathtt{A}[i] \,\}, \{\, \mathtt{B}[i] \,\}, \{\, \mathtt{C}[i,j] \,\}$

Schedule constraints

$\mathtt{A}[i] \to \mathtt{C}[i,0]$	$i \to i$	$0 \to 0$
$\mathtt{B}[i] \to \mathtt{C}[0,i]$	$0 \to 0$	$i \to i$

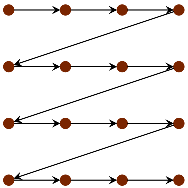
Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:   t = f(t, A[i][j]);
```



Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```



Schedule

$S[i,j] \rightarrow i$

$S[i,j] \rightarrow j$

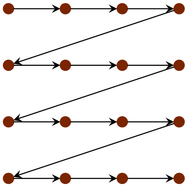
Schedule constraints

$S[i,j] \rightarrow S[i,j+1]$

$S[i,n-1] \rightarrow S[i+1,0]$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```



Schedule

$S[i,j] \to i$

$S[i,j] \to j$

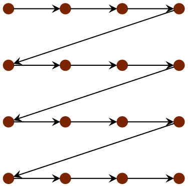
Schedule constraints

$S[i,j] \to S[i,j+1]$

$S[i,n-1] \to S[i+1,0]$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```



Schedule

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

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

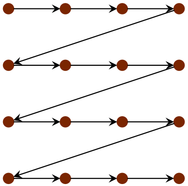
Schedule constraints

$\mathrm{S}[i,j] \to \mathrm{S}[i,j+1] \qquad i \to i$

$\mathrm{S}[i,n-1] \to \mathrm{S}[i+1,0] \qquad i \to i+1$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:   t = f(t, A[i][j]);
```



Schedule

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

$S[i,j] \to j$

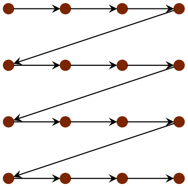
Schedule constraints

$S[i,j] \to S[i,j+1] \qquad i \to i$

$S[i,n-1] \to S[i+1,0] \qquad i \to i+1$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```



Schedule

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

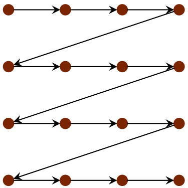
$S[i,j] \to j$

Schedule constraints

$S[i,j] \to S[i,j+1]$	$i \to i$	$j \to j+1$
$S[i,n-1] \to S[i+1,0]$	$i \to i+1$	$n-1 \to 0$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:   t = f(t, A[i][j]);
```



Schedule

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

$S[i,j] \to j$

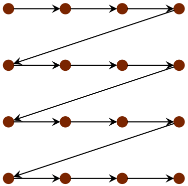
Schedule constraints

$S[i,j] \to S[i,j+1]$ $\quad i \to i \quad j \to j+1$

$S[i,n-1] \to S[i+1,0]$ $\quad i \to i+1 \quad \boxed{n-1 \to 0}$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```



Schedule

$S[i,j] \to i$

$S[i,j] \to j$

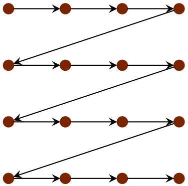
Schedule constraints

$S[i,j] \to S[i,j+1] \qquad i \to i$

$S[i,n-1] \to S[i+1,0] \qquad i \to i+1$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```



Schedule

$S[i,j] \to i$

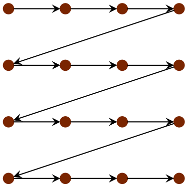
$S[i,j] \to j$

Schedule constraints

$S[i,j] \to S[i,j+1]$ $\quad i \to i$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:   t = f(t, A[i][j]);
```



Schedule

$S[i,j] \to i$

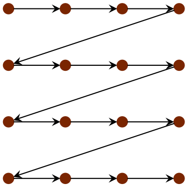
$S[i,j] \to j$

Schedule constraints

$S[i,j] \to S[i,j+1]$ $\quad i \to i$

Scheduling Example 2

```
for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```



Schedule

$S[i,j] \to i$

$S[i,j] \to j$

Schedule constraints

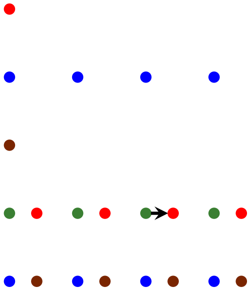
$S[i,j] \to S[i,j+1]$ $\quad i \to i \quad j \to j+1$

Relaxed Permutability Criterion

- Adjacency
  An anti-dependence is *adjacent* to a live-range
  if the source of one is the sink of the other

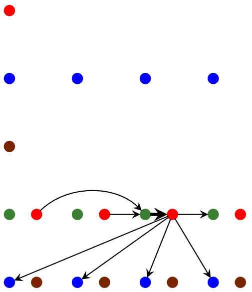
Relaxed Permutability Criterion

- Adjacency
  An anti-dependence is *adjacent* to a live-range if the source of one is the sink of the other



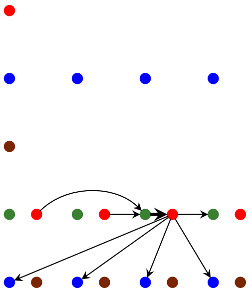
Relaxed Permutability Criterion

- Adjacency
  An anti-dependence is *adjacent* to a live-range if the source of one is the sink of the other



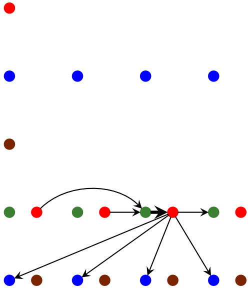
Relaxed Permutability Criterion

- Adjacency
  An anti-dependence is *adjacent* to a live-range if the source of one is the sink of the other
- Local live-ranges
  A live-range is *local* to a band if its source and sink are assigned the same value by all affine functions in the band



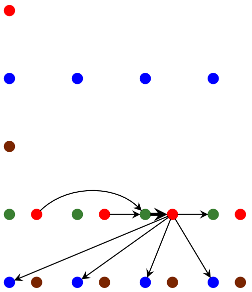
Relaxed Permutability Criterion

- Adjacency
  An anti-dependence is *adjacent* to a live-range if the source of one is the sink of the other
- Local live-ranges
  A live-range is *local* to a band if its source and sink are assigned the same value by all affine functions in the band
- Relaxed permutability criterion
  If an anti-dependence is only adjacent to live-ranges that are local to a band,
  then the anti-dependence can be ignored within the band



Relaxed Permutability Criterion

- Adjacency
  An anti-dependence is *adjacent* to a live-range if the source of one is the sink of the other
- Local live-ranges
  A live-range is *local* to a band if its source and sink are assigned the same value by all affine functions in the band
- Relaxed permutability criterion
  If an anti-dependence is only adjacent to live-ranges that are local to a band, then the anti-dependence can be ignored within the band



Baghdadi, Cohen, et al. (2013) use criterion to *reinterpret* schedule

⇒ combine nested sequences of bands after schedule construction

Conditional Validity Constraints

- A conditional validity constraint is a pair of
  – condition $\rightarrow$ live-ranges
  – conditioned validity constraint $\rightarrow$ anti-dependences

Conditional Validity Constraints

- A conditional validity constraint is a pair of
  - condition $\rightarrow$ live-ranges
  - conditioned validity constraint $\rightarrow$ anti-dependences
- A conditional validity constraint is satisfied if
  - source and sink of condition are assigned the same value, $\rightarrow$ local live-ranges

    or
  - adjacent conditional validity constraints are satisfied $\rightarrow$ adjacent anti-dependences

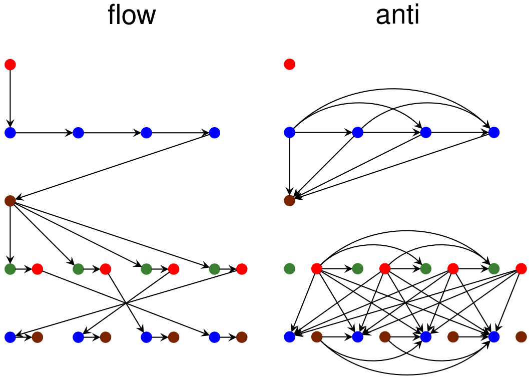
Conditional Validity Constraints

- A conditional validity constraint is a pair of
  - – condition → live-ranges
  - – conditioned validity constraint → anti-dependences
- A conditional validity constraint is satisfied if
  - – source and sink of condition are assigned the same value, → local live-ranges

    or
  - – adjacent conditional validity constraints are satisfied → adjacent anti-dependences
- Conditional validity constraints handled during schedule construction
  - ignore conditioned validity constraints during band member computation
  - compute violated conditioned validity constraints
  - compute adjacent conditions
  - force adjacent conditions to be local in subsequent band members
  - recompute band if not local in current or previous members

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{ S0[]; S1[i]; S2[] \}, \{ S3[i]; S4[i]; S5[i]; S6[i] \}$

$S0[] \to 0; S1[i] \to i; S2[] \to N-1$

$S3[i] \to i; S5[i] \to N-1-i;$

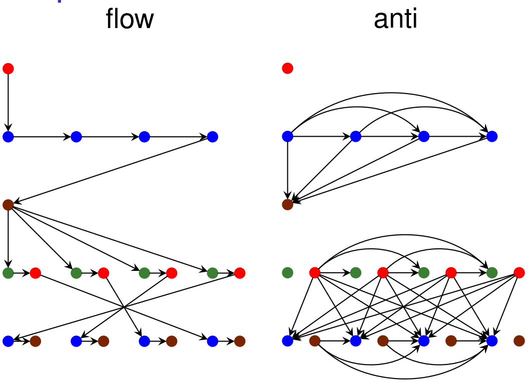
$S4[i] \to i; S6[i] \to N-1-i$

$\{ S0[] \}, \{ S1[i] \}, \{ S2[] \}$

$\{ S3[i] \}, \{ S4[i] \}, \{ S5[i] \}, \{ S6[i] \}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



{ S0[]; S1[i]; S2[] }, { S3[i]; S4[i]; S5[i]; S6[i] }

S0[] → 0; S1[i] → i; S2[] → $N-1$

S3[i] → i; S5[i] → $N-1-i$;

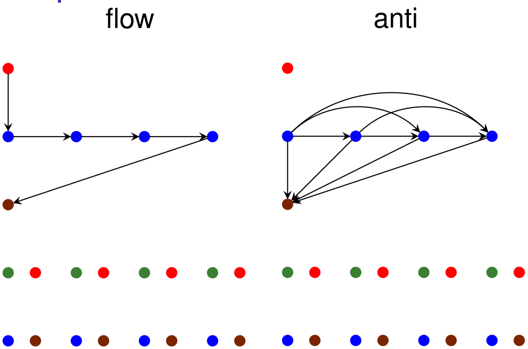
S4[i] → i; S6[i] → $N-1-i$

{ S0[] }, { S1[i] }, { S2[] }

{ S3[i] }, { S4[i] }, { S5[i] }, { S6[i] }

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{ S0[]; S1[i]; S2[] \}, \{ S3[i]; S4[i]; S5[i]; S6[i] \}$

$S0[] \to 0; S1[i] \to i; S2[] \to N - 1$

$S3[i] \to i; S5[i] \to N - 1 - i;$

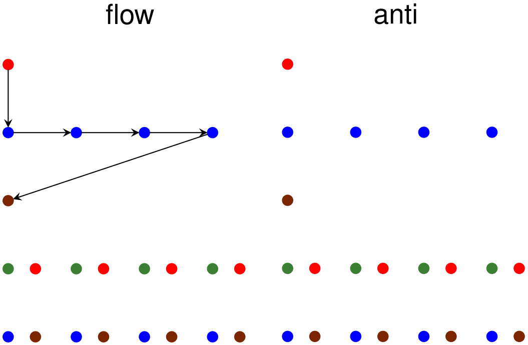
$S4[i] \to i; S6[i] \to N - 1 - i$

$\{ S0[] \}, \{ S1[i] \}, \{ S2[] \}$

$\{ S3[i] \}, \{ S4[i] \}, \{ S5[i] \}, \{ S6[i] \}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\,\mathtt{S0}[\,];\mathtt{S1}[i];\mathtt{S2}[\,]\,\},\{\,\mathtt{S3}[i];\mathtt{S4}[i];\mathtt{S5}[i];\mathtt{S6}[i]\,\}$

$\mathtt{S0}[\,]\to 0;\mathtt{S1}[i]\to i;\mathtt{S2}[\,]\to N-1$

$\mathtt{S3}[i]\to i;\mathtt{S5}[i]\to N-1-i;$

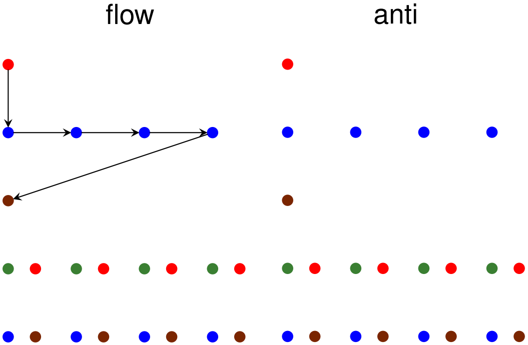
$\mathtt{S4}[i]\to i;\mathtt{S6}[i]\to N-1-i$

$\{\,\mathtt{S0}[\,]\,\},\{\,\mathtt{S1}[i]\,\},\{\,\mathtt{S2}[\,]\,\}$

$\{\,\mathtt{S3}[i]\,\},\{\,\mathtt{S4}[i]\,\},\{\,\mathtt{S5}[i]\,\},\{\,\mathtt{S6}[i]\,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



{ S0[]; S1[i]; S2[] }, { S3[i]; S4[i]; S5[i]; S6[i] }

S0[] $\to 0$; S1[i] $\to i$; S2[] $\to N - 1$

S3[i] $\to i$; S5[i] $\to N - 1 - i$;

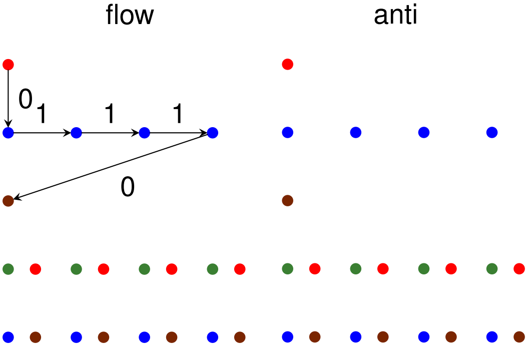
S4[i] $\to i$; S6[i] $\to N - 1 - i$

{ S0[] }, { S1[i] }, { S2[] }

{ S3[i] }, { S4[i] }, { S5[i] }, { S6[i] }

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\, \mathtt{S0}[];\ \mathtt{S1}[i];\ \mathtt{S2}[] \,\}, \{\, \mathtt{S3}[i];\ \mathtt{S4}[i];\ \mathtt{S5}[i];\ \mathtt{S6}[i] \,\}$

$\mathtt{S0}[] \to 0;\ \mathtt{S1}[i] \to i;\ \mathtt{S2}[] \to N - 1$

$\mathtt{S3}[i] \to i;\ \mathtt{S5}[i] \to N - 1 - i;$

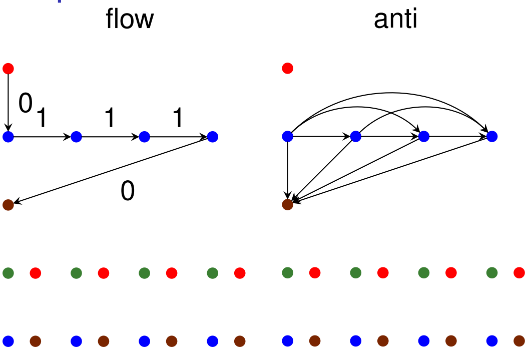
$\mathtt{S4}[i] \to i;\ \mathtt{S6}[i] \to N - 1 - i$

$\{\, \mathtt{S0}[] \,\}, \{\, \mathtt{S1}[i] \,\}, \{\, \mathtt{S2}[] \,\}$

$\{\, \mathtt{S3}[i] \,\}, \{\, \mathtt{S4}[i] \,\}, \{\, \mathtt{S5}[i] \,\}, \{\, \mathtt{S6}[i] \,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
    avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
    tmp = A[i] - avg;
    A[i] = tmp;
}
for (i=0; i<N; ++i) {
    tmp = A[N - 1 - i];
    B[i] = tmp;
}
```



$\{\, S0[];\, S1[i];\, S2[] \,\}, \{\, S3[i];\, S4[i];\, S5[i];\, S6[i] \,\}$

$S0[] \to 0;\, S1[i] \to i;\, S2[] \to N-1$

$S3[i] \to i;\, S5[i] \to N-1-i;$

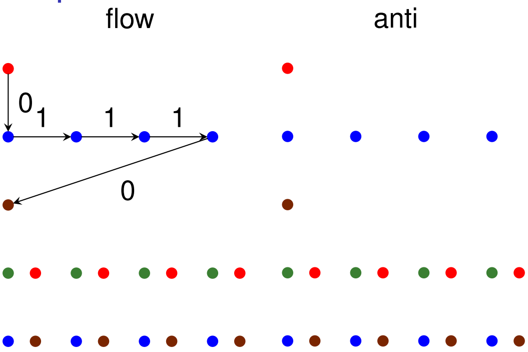
$S4[i] \to i;\, S6[i] \to N-1-i$

$\{\, S0[] \,\}, \{\, S1[i] \,\}, \{\, S2[] \,\}$

$\{\, S3[i] \,\}, \{\, S4[i] \,\}, \{\, S5[i] \,\}, \{\, S6[i] \,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{$ S0[]; S1[i]; S2[] $\}$, $\{$ S3[i]; S4[i]; S5[i]; S6[i] $\}$

S0[] $\to 0$; S1[i] $\to i$; S2[] $\to N - 1$

S3[i] $\to i$; S5[i] $\to N - 1 - i$;

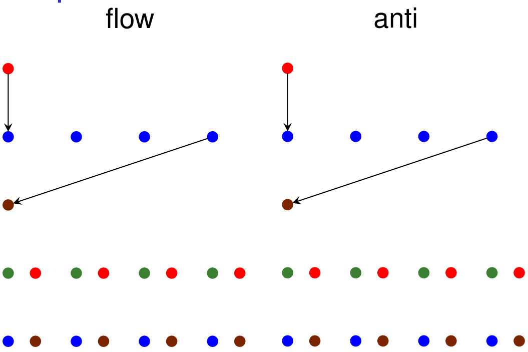
S4[i] $\to i$; S6[i] $\to N - 1 - i$

$\{$ S0[] $\}$, $\{$ S1[i] $\}$, $\{$ S2[] $\}$

$\{$ S3[i] $\}$, $\{$ S4[i] $\}$, $\{$ S5[i] $\}$, $\{$ S6[i] $\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
    avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
    tmp = A[i] - avg;
    A[i] = tmp;
}
for (i=0; i<N; ++i) {
    tmp = A[N - 1 - i];
    B[i] = tmp;
}
```



$\{ S0[]; S1[i]; S2[] \}, \{ S3[i]; S4[i]; S5[i]; S6[i] \}$

$S0[] \to 0; S1[i] \to i; S2[] \to N-1$

$S3[i] \to i; S5[i] \to N-1-i;$

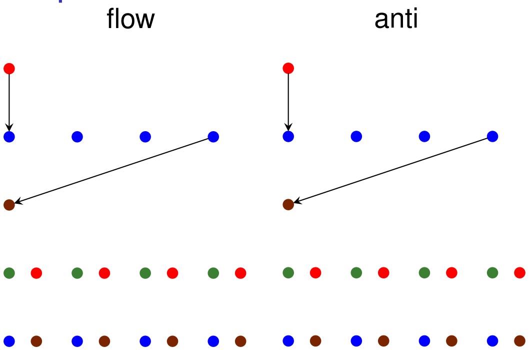
$S4[i] \to i; S6[i] \to N-1-i$

$\{ S0[] \}, \{ S1[i] \}, \{ S2[] \}$

$\{ S3[i] \}, \{ S4[i] \}, \{ S5[i] \}, \{ S6[i] \}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\, \mathtt{S0}[]; \mathtt{S1}[i]; \mathtt{S2}[] \,\}, \{\, \mathtt{S3}[i]; \mathtt{S4}[i]; \mathtt{S5}[i]; \mathtt{S6}[i] \,\}$

$\mathtt{S0}[] \to 0; \mathtt{S1}[i] \to i; \mathtt{S2}[] \to N-1$

$\mathtt{S3}[i] \to i; \mathtt{S5}[i] \to N-1-i;$

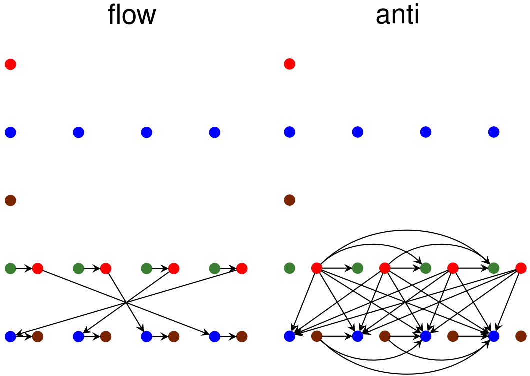
$\mathtt{S4}[i] \to i; \mathtt{S6}[i] \to N-1-i$

$\{\, \mathtt{S0}[] \,\}, \{\, \mathtt{S1}[i] \,\}, \{\, \mathtt{S2}[] \,\}$

$\{\, \mathtt{S3}[i] \,\}, \{\, \mathtt{S4}[i] \,\}, \{\, \mathtt{S5}[i] \,\}, \{\, \mathtt{S6}[i] \,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{ \mathtt{S0}[]; \mathtt{S1}[i]; \mathtt{S2}[] \}, \{ \mathtt{S3}[i]; \mathtt{S4}[i]; \mathtt{S5}[i]; \mathtt{S6}[i] \}$

$\mathtt{S0}[] \to 0; \mathtt{S1}[i] \to i; \mathtt{S2}[] \to N - 1$

$\mathtt{S3}[i] \to i; \mathtt{S5}[i] \to N - 1 - i;$

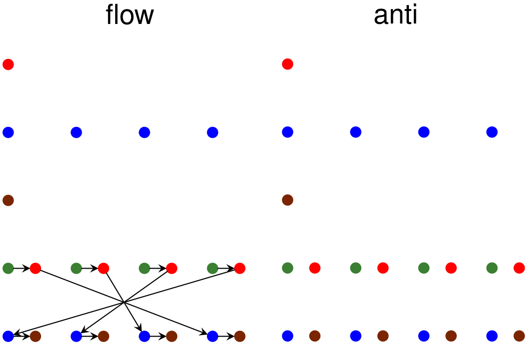
$\mathtt{S4}[i] \to i; \mathtt{S6}[i] \to N - 1 - i$

$\{ \mathtt{S0}[] \}, \{ \mathtt{S1}[i] \}, \{ \mathtt{S2}[] \}$

$\{ \mathtt{S3}[i] \}, \{ \mathtt{S4}[i] \}, \{ \mathtt{S5}[i] \}, \{ \mathtt{S6}[i] \}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\, \mathtt{S0}[]; \mathtt{S1}[i]; \mathtt{S2}[] \,\}, \{\, \mathtt{S3}[i]; \mathtt{S4}[i]; \mathtt{S5}[i]; \mathtt{S6}[i] \,\}$

$\mathtt{S0}[] \to 0; \mathtt{S1}[i] \to i; \mathtt{S2}[] \to N - 1$

$\mathtt{S3}[i] \to i; \mathtt{S5}[i] \to N - 1 - i;$

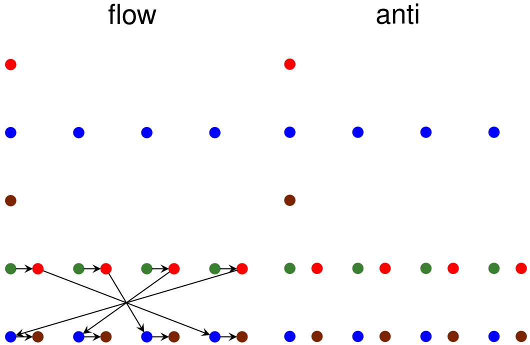
$\mathtt{S4}[i] \to i; \mathtt{S6}[i] \to N - 1 - i$

$\{\, \mathtt{S0}[] \,\}, \{\, \mathtt{S1}[i] \,\}, \{\, \mathtt{S2}[] \,\}$

$\{\, \mathtt{S3}[i] \,\}, \{\, \mathtt{S4}[i] \,\}, \{\, \mathtt{S5}[i] \,\}, \{\, \mathtt{S6}[i] \,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\, \mathtt{S0}[]; \mathtt{S1}[i]; \mathtt{S2}[] \,\}, \{\, \mathtt{S3}[i]; \mathtt{S4}[i]; \mathtt{S5}[i]; \mathtt{S6}[i] \,\}$

$\mathtt{S0}[] \to 0; \mathtt{S1}[i] \to i; \mathtt{S2}[] \to N - 1$

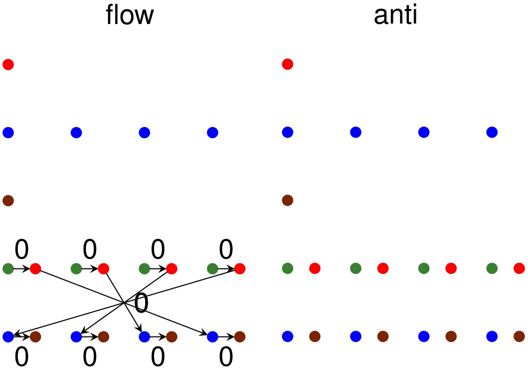
$\mathtt{S3}[i] \to i; \mathtt{S5}[i] \to N - 1 - i;$
$\mathtt{S4}[i] \to i; \mathtt{S6}[i] \to N - 1 - i$

$\{\, \mathtt{S0}[] \,\}, \{\, \mathtt{S1}[i] \,\}, \{\, \mathtt{S2}[] \,\}$

$\{\, \mathtt{S3}[i] \,\}, \{\, \mathtt{S4}[i] \,\}, \{\, \mathtt{S5}[i] \,\}, \{\, \mathtt{S6}[i] \,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\,\mathtt{S0}[];\mathtt{S1}[i];\mathtt{S2}[]\,\},\{\,\mathtt{S3}[i];\mathtt{S4}[i];\mathtt{S5}[i];\mathtt{S6}[i]\,\}$

$\mathtt{S0}[] \to 0; \mathtt{S1}[i] \to i; \mathtt{S2}[] \to N-1$

$\mathtt{S3}[i] \to i; \mathtt{S5}[i] \to N-1-i;$

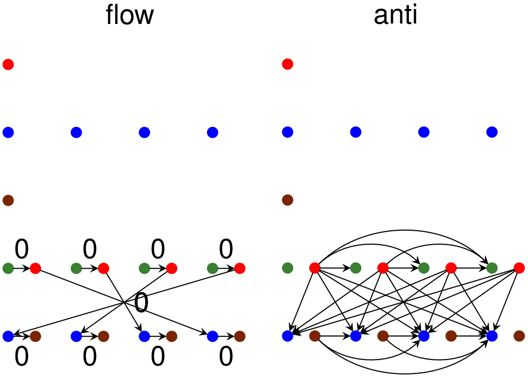
$\mathtt{S4}[i] \to i; \mathtt{S6}[i] \to N-1-i$

$\{\,\mathtt{S0}[]\,\},\{\,\mathtt{S1}[i]\,\},\{\,\mathtt{S2}[]\,\}$

$\{\,\mathtt{S3}[i]\,\},\{\,\mathtt{S4}[i]\,\},\{\,\mathtt{S5}[i]\,\},\{\,\mathtt{S6}[i]\,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\,\mathtt{S0}[];\mathtt{S1}[i];\mathtt{S2}[]\,\},\{\,\mathtt{S3}[i];\mathtt{S4}[i];\mathtt{S5}[i];\mathtt{S6}[i]\,\}$

$\mathtt{S0}[] \to 0; \mathtt{S1}[i] \to i; \mathtt{S2}[] \to N-1$

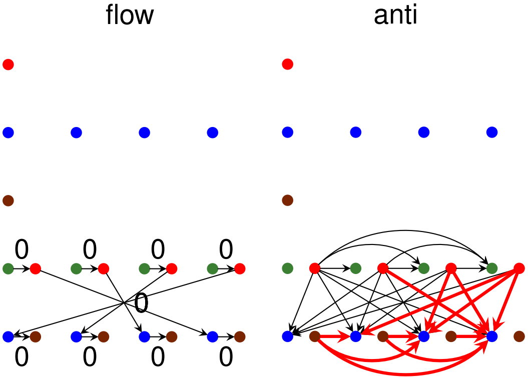
$\mathtt{S3}[i] \to i; \mathtt{S5}[i] \to N-1-i;$
$\mathtt{S4}[i] \to i; \mathtt{S6}[i] \to N-1-i$

$\{\,\mathtt{S0}[]\,\},\{\,\mathtt{S1}[i]\,\},\{\,\mathtt{S2}[]\,\}$

$\{\,\mathtt{S3}[i]\,\},\{\,\mathtt{S4}[i]\,\},\{\,\mathtt{S5}[i]\,\},\{\,\mathtt{S6}[i]\,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\, \mathtt{S0}[]; \mathtt{S1}[i]; \mathtt{S2}[] \,\}, \{\, \mathtt{S3}[i]; \mathtt{S4}[i]; \mathtt{S5}[i]; \mathtt{S6}[i] \,\}$

$\mathtt{S0}[] \to 0; \mathtt{S1}[i] \to i; \mathtt{S2}[] \to N - 1$

$\mathtt{S3}[i] \to i; \mathtt{S5}[i] \to N - 1 - i;$

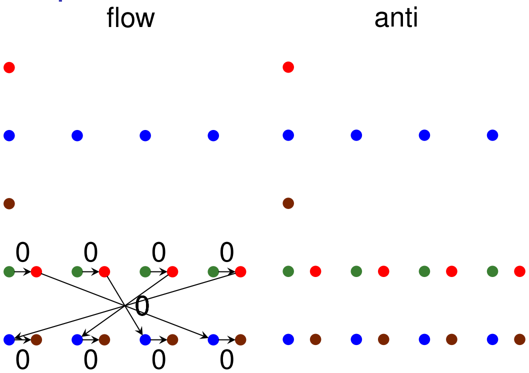
$\mathtt{S4}[i] \to i; \mathtt{S6}[i] \to N - 1 - i$

$\{\, \mathtt{S0}[] \,\}, \{\, \mathtt{S1}[i] \,\}, \{\, \mathtt{S2}[] \,\}$

$\{\, \mathtt{S3}[i] \,\}, \{\, \mathtt{S4}[i] \,\}, \{\, \mathtt{S5}[i] \,\}, \{\, \mathtt{S6}[i] \,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
   avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
   tmp = A[i] - avg;
   A[i] = tmp;
}
for (i=0; i<N; ++i) {
   tmp = A[N - 1 - i];
   B[i] = tmp;
}
```



{ S0[]; S1[i]; S2[] }, { S3[i]; S4[i]; S5[i]; S6[i] }

S0[] $\rightarrow 0$; S1[i] $\rightarrow i$; S2[] $\rightarrow N-1$

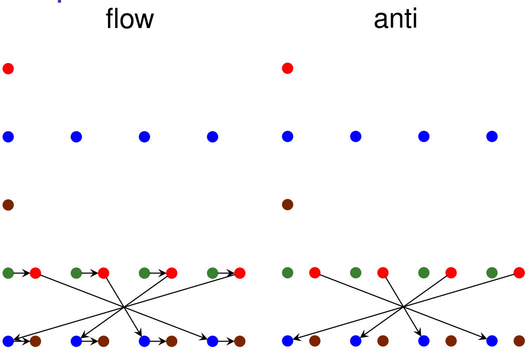
S3[i] $\rightarrow i$; S5[i] $\rightarrow N-1-i$;
S4[i] $\rightarrow i$; S6[i] $\rightarrow N-1-i$

{ S0[] }, { S1[i] }, { S2[] }

{ S3[i] }, { S4[i] }, { S5[i] }, { S6[i] }

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\, S0[\,];\, S1[i];\, S2[\,] \,\}, \{\, S3[i];\, S4[i];\, S5[i];\, S6[i] \,\}$

$S0[\,] \to 0;\, S1[i] \to i;\, S2[\,] \to N - 1$

$S3[i] \to i;\, S5[i] \to N - 1 - i;$

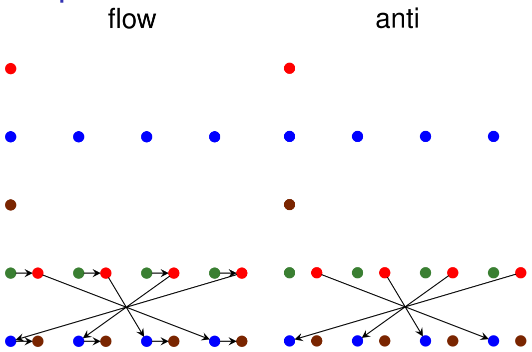
$S4[i] \to i;\, S6[i] \to N - 1 - i$

$\{\, S0[\,] \,\}, \{\, S1[i] \,\}, \{\, S2[\,] \,\}$

$\{\, S3[i] \,\}, \{\, S4[i] \,\}, \{\, S5[i] \,\}, \{\, S6[i] \,\}$

Schedule Constraints Example

```
avg = 0.f;
for (i=0; i<N; ++i)
  avg += A[i];
avg /= N;
for (i=0; i<N; ++i) {
  tmp = A[i] - avg;
  A[i] = tmp;
}
for (i=0; i<N; ++i) {
  tmp = A[N - 1 - i];
  B[i] = tmp;
}
```



$\{\, S0[]; S1[i]; S2[] \,\}, \{\, S3[i]; S4[i]; S5[i]; S6[i] \,\}$

$S0[] \to 0; S1[i] \to i; S2[] \to N-1$

$S3[i] \to i; S5[i] \to N-1-i;$

$S4[i] \to i; S6[i] \to N-1-i$

$\{\, S0[] \,\}, \{\, S1[i] \,\}, \{\, S2[] \,\}$

$\{\, S3[i] \,\}, \{\, S4[i] \,\}, \{\, S5[i] \,\}, \{\, S6[i] \,\}$

External Live-Ranges and Output Dependences

- External live-ranges
  - live-in reads

    $\Rightarrow$ order before all (later) writes
  - live-out writes

    $\Rightarrow$ order after all (earlier) reads

External Live-Ranges and Output Dependences

- External live-ranges
  - live-in reads
    - $\Rightarrow$ order before all (later) writes
  - live-out writes
    - $\Rightarrow$ order after all (earlier) reads
- Output dependences
  - there is a read between the two writes
    - $\Rightarrow$ covered by live-range and anti-dependence
  - the two writes form live-ranges with the same read
    - $\Rightarrow$ preserve order of the writes
  - first write does not appear in a live-range
    - $\Rightarrow$ add output dependence to conditioned validity constraints

Outline

Conclusion

- Enforcing anti-dependences limits scheduling freedom
- Live-range reordering
  - allows anti-dependences to be partly ignored
  - without increasing memory requirements
  - with limited loss of scheduling freedom
- Conditional validity constraints
  - allow live-range reordering during construction of schedule bands
  - available in `PPCG` since version 0.02 (April 2014)
  - crucial for experiments of Baghdadi, Beaugnon, et al. (2015)

Thanks to

- European FP7 project CARP id. 287767
- COPCAMS ARTEMIS project
- Baghdadi, Beaugnon, et al. (2015)

References I

Baghdadi, Riyadh (Sept. 2011). "Using live range non-interference constraints to enable polyhedral loop transformations". MA thesis. University of Pierre et Marie Curie - Paris 6.

Baghdadi, Riyadh, Ulysse Beaugnon, Albert Cohen, Tobias Grosser, Michael Kruse, Chandan Reddy, Sven Verdoolaege, Javed Absar, Sven van Haastregt, Alexey Kravets, Anton Lokhmotov, Adam Betts, Alastair F. Donaldson, Jeroen Ketema, Róbert Dávid, and Elnar Hajiyev (Oct. 2015). "PENCIL: A Platform-Neutral Compute Intermediate Language for Accelerator Programming". In: *Proc. Parallel Architectures and Compilation Techniques (PACT'15)*.

Baghdadi, Riyadh, Albert Cohen, Sven Verdoolaege, and Konrad Trifunovic (2013). "Improved loop tiling based on the removal of spurious false dependences". In: *TACO* 9.4, p. 52. DOI: 10.1145/2400682.2400711.

References II

Mehta, Sanyam (Sept. 2014). “Scalable Compiler Optimizations for Improving the Memory System Performance in Multi-and Many-core Processors”. PhD thesis. University of Minnesota.