Live-Range Reordering

Sven Verdoolaege\textsuperscript{1} \hspace{1cm} Albert Cohen\textsuperscript{2}

\textsuperscript{1}Polly Labs and KU Leuven
\textsuperscript{2}INRIA and École Normale Supérieure

January 19, 2016
Outline

1 Introduction
   - Example
   - Schedule Constraints

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

3 Conclusion
Outline

1. Introduction
   - Example
   - Schedule Constraints

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

3. Conclusion
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

→: execution order
Tiling Example

```c
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

```c
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:

```c
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] * 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 * B[c0 + c2][c1 + c3]) * A[c0 + c2][c4]) + (alpha * temp2));
            }
```
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 $b$
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*
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

```c
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

```c
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;
}
```

flow

---

antecedent

S0 [], S1 [i], S2 [], S3 [i], S4 [i], S5 [i], S6 [i]

S0 [] → 0;
S1 [i] → i;
S2 [] → N - 1 - i;
S3 [i] → i;
S4 [i] → i;
S5 [i] → N - 1 - i;
S6 [i]
Schedule Constraints Example

```c
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

```c
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

```c
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

```c
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

```c
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
```

Such serializing anti-dependences are very common in practice
⇒ occur in nearly all experiments of Baghdadi, Beaugnon, et al. (2015)
⇒ 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

```c
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

```c
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;
    }
```

After expansion:

```c
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

```c
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

```c
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

```c
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 $f$
  
  $f(i) < f(j) \Rightarrow i$ executed before $j$

- finite sequence $S_1, S_2, \ldots, S_n$
  
  $i \in S_{k_1} \land j \in S_{k_2} \land k_1 < k_2 \Rightarrow i$ executed before $j$
Scheduling

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

- affine functions $f$
  
  \[ f(i) < f(j) \quad \Rightarrow \quad i \text{ executed before } j \]

- finite sequence $S_1, S_2, \ldots, S_n$
  
  \[ i \in S_{k_1} \land j \in S_{k_2} \land k_1 < k_2 \quad \Rightarrow \quad i \text{ executed before } j \]

Scheduling determines schedule compatible with schedule constraints

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

\[ \Rightarrow \quad \text{there is some node with} \quad f(a) < f(b) \quad \text{or} \quad a \in S_{k_1} \land b \in S_{k_2} \land k_1 < k_2 \]

\[ \Rightarrow \quad \text{for all outer nodes} \quad f(a) = f(b) \quad \text{or} \quad \exists k : \{a, b\} \subseteq S_k \]
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
  \[ f(i) < f(j) \quad \Rightarrow \quad i \text{ executed before } j \]
- finite sequence $S_1, S_2, \ldots, S_n$
  \[ i \in S_{k_1} \land j \in S_{k_2} \land k_1 < k_2 \quad \Rightarrow \quad i \text{ executed before } j \]

Scheduling determines schedule compatible with schedule constraints

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

\[ \Rightarrow \quad \text{there is some node with} \]
\[ f(a) < f(b) \quad \text{or} \quad a \in S_{k_1} \land b \in S_{k_2} \land k_1 < k_2 \]
\[ \Rightarrow \quad \text{for all outer nodes} \]
\[ f(a) = f(b) \quad \text{or} \quad \exists k : \{a, b\} \subseteq S_k \]

*Bands*: nested sequence of affine functions that can be freely reordered
Scheduling Example 1

\begin{verbatim}
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]);
\end{verbatim}
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

\[
A[i] \rightarrow i; B[i] \rightarrow 0; C[i, j] \rightarrow i
\]

Schedule constraints

\[
A[i] \rightarrow C[i, 0];
B[i] \rightarrow C[0, i];
C[i, j] \rightarrow C[i + 1, j];
C[i, j] \rightarrow C[i, j + 1];
\]
Scheduling Example 1

\begin{align*}
\textbf{for} & \ (i = 1; \ i < n; \ ++i) \\
\textbf{A:} & M[i, \ 0] = f(); \\
\textbf{for} & \ (i = 1; \ i < n; \ ++i) \\
\textbf{B:} & M[0, \ i] = g(); \\
\textbf{for} & \ (i = 1; \ i < n; \ ++i) \\
\ & \quad \textbf{for} \ (j = 1; \ j < n; \ ++j) \\
\textbf{C:} & M[i][j] = h(M[i-1][j], \ M[i][j-1]);
\end{align*}

Schedule
\begin{align*}
A[i] & \rightarrow i; \ B[i] \rightarrow 0; \ C[i, j] \rightarrow i \\
\{A[i]\}, \{B[i]\}, \{C[i, j]\}
\end{align*}

Schedule constraints
\begin{align*}
A[i] & \rightarrow C[i, 0] \\
B[i] & \rightarrow C[0, i] \\
C[i, j] & \rightarrow C[i + 1, j] \\
C[i, j] & \rightarrow C[i, j + 1]
\end{align*}
Scheduling Example 1

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

Schedule
\begin{align*}
A[i] & \rightarrow i; B[i] \rightarrow 0; C[i, j] \rightarrow i \\
\{ A[i] \}, \{ B[i] \}, \{ C[i, j] \}
\end{align*}

Schedule constraints
\begin{align*}
A[i] & \rightarrow C[i, 0] & i & \rightarrow i \\
B[i] & \rightarrow C[0, i] & 0 & \rightarrow 0 \\
C[i, j] & \rightarrow C[i + 1, j] & i & \rightarrow i + 1 \\
C[i, j] & \rightarrow C[i, j + 1] & i & \rightarrow i
\end{align*}
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

\[
\begin{align*}
A[i] &\rightarrow i; B[i] \rightarrow 0; C[i, j] \rightarrow i \\
A[i] &\rightarrow 0; B[i] \rightarrow i; C[i, j] \rightarrow j \\
\end{align*}
\]

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

Schedule constraints

\[
\begin{align*}
A[i] &\rightarrow C[i, 0] & i \rightarrow i \\
B[i] &\rightarrow C[0, i] & 0 \rightarrow 0 \\
C[i, j] &\rightarrow C[i + 1, j] & i \rightarrow i + 1 \\
C[i, j] &\rightarrow C[i, j + 1] & i \rightarrow i \\
\end{align*}
\]
Scheduling Example 1

\[
\begin{align*}
&\textbf{for } (i = 1; i < n; ++i) \\
&A: M[i, 0] = f(); \\
&B: M[0, i] = g(); \\
&\textbf{for } (i = 1; i < n; ++i) \\
&\quad \textbf{for } (j = 1; j < n; ++j) \\
&C: M[i][j] = h(M[i-1][j], M[i][j-1]);
\end{align*}
\]

Schedule
\[
\begin{align*}
&A[i] \rightarrow i; B[i] \rightarrow 0; C[i, j] \rightarrow i \\
&A[i] \rightarrow 0; B[i] \rightarrow i; C[i, j] \rightarrow j \\
\{ A[i] \}, \{ B[i] \}, \{ C[i, j] \}
\end{align*}
\]

Schedule constraints
\[
\begin{align*}
&A[i] \rightarrow C[i, 0] & i \rightarrow i & 0 \rightarrow 0 \\
&B[i] \rightarrow C[0, i] & 0 \rightarrow 0 & i \rightarrow i \\
&C[i, j] \rightarrow C[i + 1, j] & i \rightarrow i + 1 & j \rightarrow j \\
&C[i, j] \rightarrow C[i, j + 1] & i \rightarrow i & j \rightarrow j + 1
\end{align*}
\]
Scheduling Example 1

\begin{verbatim}
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]);
\end{verbatim}

Schedule

\begin{align*}
\text{A}[i] & \rightarrow i; \text{B}[i] \rightarrow 0; \text{C}[i, j] \rightarrow i \\
\text{A}[i] & \rightarrow 0; \text{B}[i] \rightarrow i; \text{C}[i, j] \rightarrow j
\end{align*}

Schedule constraints

\begin{align*}
\text{A}[i] & \rightarrow \text{C}[i, 0] & i & \rightarrow i & 0 & \rightarrow 0 \\
\text{B}[i] & \rightarrow \text{C}[0, i] & 0 & \rightarrow 0 & i & \rightarrow i
\end{align*}
Scheduling Example 1

```c
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

- \(A[i] \rightarrow i\)
- \(B[i] \rightarrow 0\)
- \(C[i, j] \rightarrow i\)
- \(A[i] \rightarrow 0\)
- \(B[i] \rightarrow i\)
- \(C[i, j] \rightarrow j\)

Schedule constraints

- \(A[i] \rightarrow C[i, 0]\)
- \(i \rightarrow i\)
- \(0 \rightarrow 0\)
- \(B[i] \rightarrow C[0, i]\)
- \(0 \rightarrow 0\)
- \(i \rightarrow i\)
Scheduling Example 2

```c
for (i = 0; i < n; ++i)
    for (j = 0; j < n; ++j)
S:  t = f(t, A[i][j]);
```
Scheduling Example 2

\[
\text{for } (i = 0; i < n; ++i) \\
\quad \text{for } (j = 0; j < n; ++j) \\
S: \ t = f(t, A[i][j]);
\]

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

\[ S[i,j] \rightarrow i \]

\[ S[i,j] \rightarrow j \]

Schedule constraints

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

\[ S[i,n-1] \rightarrow S[i+1,0] \]
Scheduling Example 2

\[
\text{for } (i = 0; i < n; ++i) \\
\quad \text{for } (j = 0; j < n; ++j) \\
S: \quad t = f(t, A[i][j]);
\]

Schedule
\[
S[i, j] \rightarrow i \\
S[i, j] \rightarrow j
\]

Schedule constraints
\[
S[i, j] \rightarrow S[i, j + 1] \quad i \rightarrow i \\
S[i, n - 1] \rightarrow S[i + 1, 0] \quad i \rightarrow i + 1
\]
Scheduling Example 2

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

Schedule
\[ S[i, j] \rightarrow i, S[i, j] \rightarrow j \]
\[ S[i, j] \rightarrow j \]

Schedule constraints
\[ S[i, j] \rightarrow S[i, j + 1] \quad i \rightarrow i \]
\[ S[i, n - 1] \rightarrow S[i + 1, 0] \quad i \rightarrow i + 1 \]
Scheduling Example 2

\[
\text{for}\ (i = 0; i < n; ++i) \\
\quad \text{for}\ (j = 0; j < n; ++j) \\
S: \ t = f(t, A[i][j]);
\]

Schedule
\[
S[i, j] \rightarrow i, S[i, j] \rightarrow j \\
S[i, j] \rightarrow j
\]

Schedule constraints
\[
S[i, j] \rightarrow S[i, j + 1] \quad i \rightarrow i \quad j \rightarrow j + 1 \\
S[i, n - 1] \rightarrow S[i + 1, 0] \quad i \rightarrow i + 1 \quad n - 1 \rightarrow 0
\]
Scheduling Example 2

\[
\begin{align*}
\text{for } & (i = 0; i < n; ++i) \\
& \quad \text{for } (j = 0; j < n; ++j) \\
S: & \quad t = f(t, A[i][j]);
\end{align*}
\]

Schedule
\[
\begin{align*}
S[i, j] & \rightarrow i, S[i, j] \rightarrow j \\
S[i, j] & \rightarrow j
\end{align*}
\]

Schedule constraints
\[
\begin{align*}
S[i, j] & \rightarrow S[i, j + 1] & i & \rightarrow i \\
S[i, n - 1] & \rightarrow S[i + 1, 0] & i & \rightarrow i + 1
\end{align*}
\]
Scheduling Example 2

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

Schedule constraints

- $S[i, j] \rightarrow S[i, j + 1]$  \hspace{1cm} i \rightarrow i$
- $S[i, n - 1] \rightarrow S[i + 1, 0]$  \hspace{1cm} i \rightarrow i + 1

Schedule

- $S[i, j] \rightarrow i$
- $S[i, j] \rightarrow j$
Scheduling Example 2

```latex
\begin{align*}
\textbf{for} & \ (i = 0; i < n; \ ++i) \\
& \textbf{for} \ (j = 0; j < n; \ ++j) \\
S: & \ t = f(t, A[i][j]);
\end{align*}
```

Schedule
\begin{align*}
S[i, j] & \rightarrow i \\
S[i, j] & \rightarrow j
\end{align*}

Schedule constraints
\begin{align*}
S[i, j] & \rightarrow S[i, j + 1] \\
i & \rightarrow i
\end{align*}
Scheduling Example 2

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

Schedule constraints

```
S[i, j] → S[i, j + 1]  i → i
```

Schedule

```
S[i, j] → i
S[i, j] → j
```
Scheduling Example 2

\[
\text{for (i = 0; i < n; ++i)} \\
\hspace{1em} \text{for (j = 0; j < n; ++j)} \\
S: \ t = f(t, A[i][j]);
\]

Schedule constraints:
\[
S[i, j] \rightarrow i \\
S[i, j] \rightarrow S[i, j + 1] \\
i \rightarrow i \\
j \rightarrow j + 1
\]
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 → live-ranges
  - conditioned validity constraint → anti-dependences
Conditional Validity Constraints

- A conditional validity constraint is a pair of
  - condition
  - conditioned validity constraint
  → live-ranges
  → anti-dependences

- A conditional validity constraint is satisfied if
  - source and sink of condition
    are assigned the same value,
    or
  - adjacent conditional validity constraints are satisfied
    → local live-ranges
    → 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
  - 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

```c
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
{ S0[] }, { S1[i] }, { S2[] }  
{ S3[i] }, { S4[i] }, { S5[i] }, { S6[i] }
```
Schedule Constraints Example

```c
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
Schedule Constraints Example

```c
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
```
Schedule Constraints Example

```c
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
```
Schedule Constraints Example

```c
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;
}
```

The schedule constraints are represented by the flow and anti flows as follows:

- **Flow Flows**:
  - \( S0[] \rightarrow 0; S1[i] \rightarrow i; S2[] \rightarrow N - 1 \)

- **Anti Flows**:
  - \( S3[i] \rightarrow i; S5[i] \rightarrow N - 1 - i; S4[i] \rightarrow i; S6[i] \rightarrow N - 1 - i \)
Schedule Constraints Example

```c
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
```
Schedule Constraints Example

\[
\text{avg} = 0.0;
\]
\[
\text{for} \ (i=0; \ i<N; \ +i) \\
\quad \text{avg} += \text{A}[i];
\]
\[
\text{avg} /= N;
\]
\[
\text{for} \ (i=0; \ i<N; \ +i) \{ \\
\quad \text{tmp} = \text{A}[i] - \text{avg};
\quad \text{A}[i] = \text{tmp};
\}
\]
\[
\text{for} \ (i=0; \ i<N; \ +i) \{ \\
\quad \text{tmp} = \text{A}[N - 1 - i];
\quad \text{B}[i] = \text{tmp};
\}
\]

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

\[
\text{S0}[] \rightarrow 0; \text{S1}[i] \rightarrow i; \text{S2}[] \rightarrow N - 1
\]

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

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

\[
\text{S3}[i] \rightarrow i; \text{S5}[i] \rightarrow N - 1 - i;
\]

\[
\text{S4}[i] \rightarrow i; \text{S6}[i] \rightarrow N - 1 - i
\]
Schedule Constraints Example

```c
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;
}
```

Flow anti

- `S0[] → 0; S1[i] → i; S2[] → N - 1`
- `S3[i] → i; S5[i] → N - 1 - i; S4[i] → i; S6[i] → N - 1 - i`
Schedule Constraints Example

```c
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
Schedule Constraints Example

```c
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] }
Schedule Constraints Example

\[
\begin{align*}
\text{avg} & = 0.0f; \\
\text{for } (i=0; i<N; ++i) & \quad \text{flow} \\
& \quad \text{avg} += A[i]; \\
& \quad \text{avg} /= N; \\
\text{for } (i=0; i<N; ++i) & \quad \text{anti} \\
& \quad \text{tmp} = A[i] - \text{avg}; \\
& \quad A[i] = \text{tmp}; \\
& \{ S0[], S1[i], S2[] \}, \{ S3[i], S4[i], S5[i], S6[i] \} \\
\text{S0[]} & \rightarrow 0; \text{S1[i]} \rightarrow i; \text{S2[]} \rightarrow N-1 \\
\{ S0[] \}, \{ S1[i] \}, \{ S2[] \} \\
\text{S3[i]} & \rightarrow i; \text{S5[i]} \rightarrow N-1 - i; \\
\{ S3[i] \}, \{ S4[i] \}, \{ S5[i] \}, \{ S6[i] \}
\end{align*}
\]
Schedule Constraints Example

\[
\text{avg} = 0.f;
\]

\[
\text{for } (i=0; i<N; ++i) \\
\quad \text{avg } += \text{A}[i]; \\
\quad \text{avg } /= \text{N}; \\
\text{for } (i=0; i<N; ++i) \\
\quad \text{tmp } = \text{A}[i] - \text{avg}; \\
\quad \text{A}[i] = \text{tmp}; \\
\}
\]

\[
\text{for } (i=0; i<N; ++i) \\
\quad \text{tmp } = \text{A}[N - 1 - i]; \\
\quad \text{B}[i] = \text{tmp}; \\
\}
\]

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

\[
\text{S0[]} \rightarrow 0; \text{S1}[i] \rightarrow i; \text{S2[]} \rightarrow N - 1
\]

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

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

\[
\text{S3}[i] \rightarrow i; \text{S5}[i] \rightarrow N - 1 - i; \\
\text{S4}[i] \rightarrow i; \text{S6}[i] \rightarrow N - 1 - i
\]
Schedule Constraints Example

```c
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;
}
```

The diagram illustrates the flow and anti-flow graph for the given schedule constraints. The graph shows the data dependencies between the variables `A[i]`, `B[i]`, and `i` where `i` ranges from 0 to `N-1`. The flow graph shows the forward data dependencies, while the anti-flow graph shows the reverse dependencies.

- **Flow Graph:**
  - `S0[]` → 0
  - `S1[i]` → `i`
  - `S2[]` → `N - 1`

- **Anti-Flow Graph:**
  - `S3[i]` → `i`
  - `S5[i]` → `N - 1 - i`
  - `S4[i]` → `i`
  - `S6[i]` → `N - 1 - i`
Schedule Constraints Example

```c
float avg = 0.0f;
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;
}
```

### Flow Graph

- **S0[] → 0**
- **S1[i] → i**
- **S2[] → N - 1**
- **S3[i] → i**
- **S4[i] → i**
- **S5[i] → N - 1 - i**
- **S6[i] → N - 1 - i**

---

### Conditional Validity Constraints
Schedule Constraints Example

```c
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[j]; S2[] }, { S3[i]; S4[i]; S5[i]; S6[i] }
S0[] → 0; S1[i] → i; S2[] → N - 1
{ S0[] }, { S1[i] }, { S2[] }
{ S3[i] }, { S4[i] }, { S5[i] }, { S6[i] }
```
Schedule Constraints Example

```c
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
{ S0[] }, { S1[i] }, { S2[] }  
{ S3[i], S4[i], S5[i], S6[i] }
S3[i] → i; S5[i] → N - 1 - i;
S4[i] → i; S6[i] → N - 1 - i
```
**Schedule Constraints Example**

```c
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;
}
```

```plaintext
{ 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

```c
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

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

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

S3[i] → i; S5[i] → N - 1 - i;
S4[i] → i; S6[i] → N - 1 - i
```
Schedule Constraints Example

```c
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;
}
```
External Live-Ranges and Output Dependences

- External live-ranges
  - live-in reads
    - order before all (later) writes
  - live-out writes
    - order after all (earlier) reads
External Live-Ranges and Output Dependences

- **External live-ranges**
  - live-in reads
    - order before all (later) writes
  - live-out writes
    - order after all (earlier) reads

- **Output dependences**
  - there is a read between the two writes
    - covered by live-range and anti-dependence
  - the two writes form live-ranges with the same read
    - preserve order of the writes
  - first write does not appear in a live-range
    - add output dependence to conditioned validity constraints
Outline

1. Introduction
   - Example
   - Schedule Constraints

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

3. Conclusion
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
References I


References II