Liveness Analysis in Explicitly-Parallel Programs

Alain Darte    Alexandre Isoard    Tomofumi Yuki

Compsys, LIP, Lyon

6th International Workshop on Polyhedral Compilation Techniques
January 19, 2016
Prague, Czech Republic
Use of liveness analysis

Necessary for memory reuse:
- Register allocation: interference graph
- Array contraction: conflicting relation
- Wire usage: bitwidth analysis

Important information for:
- Communication: live-in/live-out sets (inlining, offloading)
- Memory footprint: cache prediction
- Lower/upper bounds on memory usage
Why revisit liveness analysis?

Several variants:
- Value-based or memory-based analysis
- Liveness sets or interference graphs
- Control flow graphs: basic blocks, SSA, SSI, etc...

What about task graphs? Or parallel specifications in general?
- Alpha, OpenStream
- CUDA/OpenCL
- OpenMP (loop parallelism), OpenMP 4.0 (dependent tasks)
- X10 (async, finish, clocks)
- ...
Key contribution

Liveness analysis based on “happens-before” relations.

Key remarks:

- No global notion of time
- Polyhedral fragments of OpenMP, X10, ... can be handled
- Room for approximations
Outline

- Introduction
- Recap of sequential case
- Direct extensions
- Using happens-before relation
- Some properties
- Conclusion
Register allocation

\[
x = \ldots;
y = x + \ldots;
\ldots = y;
\]

\[
x = \ldots;
x = x + \ldots;
\ldots = x;
\]

write \(x\)
read \(x\)
write \(x\)
read \(x\)
write \(x\)
read \(x\)
write \(x\)
read \(x\)
Array folding

c[0] = ...;
for (i=0; i<n; ++i)
    c[i+1] = c[i] + ...;

c = ...;
for (i=0; i<n; ++i)
    c = c + ...;

\[
\begin{align*}
\vdots \\
c[i-1] & | \text{write } c_{i-1} \\
c[i] & | \text{read } c_{i-1} \\
c[i] & | \text{write } c_i \\
\text{read } c_i \\
c[i+1] & | \text{write } c_{i+1} \\
\text{read } c_{i+1} \\
\vdots \\
\end{align*}
\]
Jacobi-1D: Sequential

```plaintext
for(i=0; j<n; ++i)
    for(j=0; j<n; ++j)
```

A[i][j] $\mapsto$ A[(j-i)%(n+1)]
Simultaneously live: “Crossproduct”

Definition (Conflict)

Two memory cells $x$ and $y$ conflicts iff there exists a time step $t$ where they are both live.

$W_x$ write of $x$

$R_x$ read of $x$

$W_y$ write of $y$

$R_y$ read of $y$

$x \triangleleft y$
Liveness at a given time step with iscc

# Inputs
Params := [n] -> { : n >= 0 };  
Domain := [n] -> { S[i,j] : 0 <= i, j < n };  
Read := [n] -> { S[i,j] -> A[i-1,j-1]; S[i,j] -> A[i-1,j];  
                S[i,j] -> A[i-1,j+1] } * Domain;  
Sched := [n] -> { S[i,j] -> [i,j] };  

# Operators
Prev := { [i,j]->[k,l]: i<k or (i=k and j<l) };  
Preveq := { [i,j]->[k,l]: i<k or (i=k and j<=l) };  
WriteBeforeTStep := (Prev^-1).(Sched^-1).Write;  
ReadAfterTStep := Preveq.(Sched^-1).Read;  

# Liveness and conflicts
Live := WriteBeforeTStep * ReadAfterTStep;  
Conflict := (Live^-1).Live;  
Delta := deltas Conflict;  

\[ \Delta(n) = \{(1, i_1) | i_1 \leq 0, n \geq 3, i_1 \geq 1 - n\} \cup \]  
\[ \{(0, i_1) | i_1 \geq 1 - n, n \geq 2, i_1 \leq -1 + n\} \cup \]  
\[ \{(-1, i_1) | i_1 \geq 0, n \geq 3, i_1 \leq -1 + n\} \]
Simultaneously live: “Triangle” (Register allocation)

**Definition (Conflict)**

Two memory cells $x$ and $y$ conflict iff one is live at a write of the other.

\[ W_x \text{ write of } x \]
\[ R_x \text{ read of } x \]
\[ t \]
\[ W_y \]
\[ \vee \text{ sym } \iff x \otimes y \]

\[ W_x \rightarrow t \]
\[ R_x \rightarrow t \]
"Crossproduct" vs "Triangle"

if(...) x = ...;
else y = ...;

if(...) ... = x;
else ... = y;

**Crossproduct**  Will detect a conflict

**Triangle**   Will not detect a conflict
“Crossproduct” vs “Triangle”

```c
if(...) x = ...;
else y = ...;

if(...) ... = x;
else ... = y;
```

**Crossproduct** Will detect a conflict

**Triangle** Will not detect a conflict

👉 Valid because no legal trace are affected
for(i=0; j<n; ++i)
#pragma omp parallel for
    for(j=0; j<n; ++j)
How general?

**Inner parallelism**  Almost the same as sequential.

**Series parallel**  Can use a careful hierarchical approach.
How general?

Inner parallelism  Almost the same as sequential.
Series parallel  Can use a careful hierarchical approach.
Software pipelining  Harder to get a concept of “time”.

\[
\begin{align*}
  S &\to L \\
  C &\to L \\
  S &\to C \\
  L &\to C \\
\end{align*}
\]
How general?

**Inner parallelism**  Almost the same as sequential.

**Series parallel**  Can use a careful hierarchical approach.

**Software pipelining**  Harder to get a concept of “time”.

\[
\begin{align*}
S(i - 1) & \bowtie C(i) \text{ and } C(i) \bowtie L(i + 1) \text{ but not } S(i - 1) \bowtie L(i + 1). \\
\text{Not a clique!}
\end{align*}
\]
Potentially simultaneously live

Definition (Conflict)
Two memory cells $x$ and $y$ conflicts iff there exists a trace where one is live at a write of the other.

Definition (Happens-before)
a happens-before $b$ iff, in all traces where $a$ and $b$ are executed, $a$ is executed before $b$.

If:
- A trace is assumed possible iff it is allowed by happens-before
- Happens-before is a partial order (transitive closure)

then:

$$\exists t : \text{trace}$$

\[
\begin{align*}
W_x & \xrightarrow{t} W_y \\
R_x & \xleftarrow{t} R_x \\
W_x & \xrightarrow{t} W_y \\
& \iff \\
& \iff x \niota y
\end{align*}
\]
Potentially simultaneously live

Definition (Conflict)
Two memory cells $x$ and $y$ conflict iff there exists a trace where one is live at a write of the other.

Definition (Happens-before)
a happens-before $b$ iff, in all traces where $a$ and $b$ are executed, $a$ is executed before $b$.

If:
- A trace is assumed possible iff it is allowed by happens-before

then:

\[
\exists t : \text{trace} \quad \iff \quad x \bowtie y
\]
Folk corollary

Corollary (when happens-before is a partial order)

A source-to-source memory transformation that respects the conflicts preserves all the parallelism captured by the happens-before relation.

\[
\begin{align*}
\text{if}(b) & \quad x = \ldots; & \text{if}(b) & \quad x = \ldots; \\
\text{if}(\text{not } b) & \quad y = \ldots; & = & \quad \text{if}(c) \quad \ldots = x; \\
\text{if}(c) & \quad \ldots = x; & \text{if}(\text{not } b) & \quad y = \ldots; \\
\text{if}(\text{not } c) & \quad \ldots = y; & \text{if}(\text{not } c) & \quad \ldots = y;
\end{align*}
\]
Folk corollary

Corollary (when happens-before is a partial order)

A source-to-source memory transformation that respects the conflicts preserves all the parallelism captured by the happens-before relation.

\[
\begin{align*}
  \text{if}(b) & \quad x = \ldots; \\
  \text{if}(\text{not } b) & \quad y = \ldots; \\
  \text{if}(c) & \quad \ldots = x; \\
  \text{if}(\text{not } c) & \quad \ldots = y;
\end{align*}
\]

\[
\begin{align*}
  W_x & \quad W_x \quad W_y \quad W_y \\
  \downarrow & \quad \downarrow \quad \downarrow \\
  R_x & \quad R_y \quad R_x \quad R_y
\end{align*}
\]
Folk corollary

Corollary (when happens-before is a partial order)
A source-to-source memory transformation that respects the conflicts preserves all the parallelism captured by the happens-before relation.

\[
\begin{align*}
if(b) & \quad x = \ldots; \\
if(\neg b) & \quad y = \ldots; \\
if(c) & \quad \ldots = x; \\
if(\neg c) & \quad \ldots = y;
\end{align*}
\]

traces: \[ W_x \quad W_x \quad W_y \quad W_y \]

happens-before: \[ R_x \quad R_y \quad R_x \quad R_y \]

\[
\begin{align*}
if(b) & \quad x = \ldots; \\
if(c) & \quad \ldots = x; \\
if(\neg b) & \quad y = \ldots; \\
if(\neg c) & \quad \ldots = y;
\end{align*}
\]
Folk corollary

**Corollary (when happens-before is a partial order)**

A source-to-source memory transformation that respects the conflicts preserves *all* the parallelism captured by the happens-before relation.

\[
\begin{align*}
\text{if}(b) & \quad x = \ldots; \\
\text{if}(\text{not } b) & \quad x = \ldots; \\
\text{if}(c) & \quad \ldots = x; \\
\text{if}(\text{not } c) & \quad \ldots = x;
\end{align*}
\]

\[
\begin{align*}
\text{traces:} & \quad W_x \quad W_x \quad W_y \quad W_y \\
\text{happens-before:} & \quad R_x \quad R_y \quad R_x \quad R_y
\end{align*}
\]
Theorem (when happens-before is a partial order)

If no dead code, no undefined read, but possibly races, the interference graph is the complement of a comparability graph: the reuse graph.

Consequences:

- Perfect graph: max color = max clique;
- Dilworth theorem: coloring polynomially computable;
- Link with “reuse graph” of work on (Q)UOV.

But not particularly useful in the polyhedral framework: would require enumeration of iterations.
Wrap-up

**Trace-independent:** if allocation respects it is valid for any trace.

**Happens-before:** quite general, handle *if conditions* (conservatively), do not handle *critical sections* (will assume possible conflict).

**Optimality:** size = max clique, polynomially computable (Dilworth) if graph is given in extension (unlike polyhedral optimization).

**Source-to-source transformation:** contraction can be expressed in the same specification model, without constraining parallelism further.
Conclusion

Possible future work:

- Critical sections are not captured by happens-before
  - hierarchical happens-before?
- Explicit handling of control directly exploiting CFG?
- Code generation from happens-before relation?

Towards a better understanding of parallel languages: semantics, static analysis, and links with the runtime.
## Buffer Sizes

<table>
<thead>
<tr>
<th>Sequential Memory Size</th>
<th>Pipelined Memory Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jacobi-1d-imper</strong></td>
<td></td>
</tr>
<tr>
<td>$A[2s_1 + s_2]$</td>
<td>$A[2s_1 + 2s_2]$</td>
</tr>
<tr>
<td>$B[2s_1 + s_2 - 1]$</td>
<td>$B[2s_1 + 2s_2 - 2]$</td>
</tr>
<tr>
<td><strong>jacobi-2d-imper</strong></td>
<td></td>
</tr>
<tr>
<td>$A[2s_1 + s_2, \min(2s_1, s_2 + 1) + s_3]$</td>
<td>$A[2s_1 + s_2, \min(2s_1, s_2 + 1) + 2s_3]$</td>
</tr>
<tr>
<td>$B[2s_1 + s_2 - 1, \min(2s_1, s_2 + 1) + s_3 - 1]$</td>
<td>$B[2s_1 + s_2 - 1, \min(2s_1, s_2 + 1) + 2s_3 - 2]$</td>
</tr>
<tr>
<td><strong>seidel-2d</strong></td>
<td></td>
</tr>
<tr>
<td>$A\left[ s_1 + s_2 + 1, \min(2s_1 + 2, s_1 + s_2, 2s_2 + 2) + s_3 \right]$</td>
<td>$A\left[ s_1 + s_2 + 1, \min(2s_1 + 2, s_1 + s_2, 2s_2 + 2) + 2s_3 \right]$</td>
</tr>
<tr>
<td><strong>gemm</strong></td>
<td></td>
</tr>
<tr>
<td>$A[s_1, s_3]$</td>
<td>$A[s_1, 2s_3]$</td>
</tr>
<tr>
<td>$B[s_3, s_2]$</td>
<td>$B[2s_3, s_2]$</td>
</tr>
<tr>
<td>$C[s_1, s_2]$</td>
<td>$C[s_1, s_2]$</td>
</tr>
<tr>
<td><strong>floyd-warshall</strong></td>
<td></td>
</tr>
<tr>
<td>$\text{path}\left[ \max(k + 1, n - k), \right]$</td>
<td>$\text{path}\left[ \max(k + 1, n - k), \right.$</td>
</tr>
<tr>
<td>$\left. \max(k + 1, n - k) \right]$</td>
<td>$\left. \max(k + 1, n - k, 2s_2) \right]$</td>
</tr>
</tbody>
</table>