Fast Profile Driven Partial Dead Code Elimination

for JIT compilers

David Pereira Nigel Horspool

{djp,nigelh}@cs.uvic.ca

University of Victoria

British Columbia

Canada

Introduction

- Dead Code Elimination (DCE) and Partial DCE:

Introduction

- Dead Code Elimination (DCE) and Partial DCE:
  - Important

Introduction

- Dead Code Elimination (DCE) and Partial DCE:
  - Important
  - Pervasive

Introduction

- Dead Code Elimination (DCE) and Partial DCE:
  - Important
  - Pervasive
- However, they are **profile-naïve**

Introduction

- Dead Code Elimination (DCE) and Partial DCE:
  - Important
  - Pervasive
- However, they are **profile-naïve**
- Profile-aware variations do exist

Introduction

- Dead Code Elimination (DCE) and Partial DCE:
  - Important
  - Pervasive
- However, they are **profile-naïve**
- Profile-aware variations do exist
- However, they are **very expensive**

Introduction

- Dead Code Elimination (DCE) and Partial DCE:
  - Important
  - Pervasive
- However, they are **profile-naïve**
- Profile-aware variations do exist
- However, they are **very expensive**
  - $O(n^3)$ for each expression

Introduction

- Dead Code Elimination (DCE) and Partial DCE:
  - Important
  - Pervasive
- However, they are **profile-naïve**
- Profile-aware variations do exist
- However, they are **very expensive**
  - $O(n^3)$ for each expression
- We need an effective **linear** approximation

Dead Code

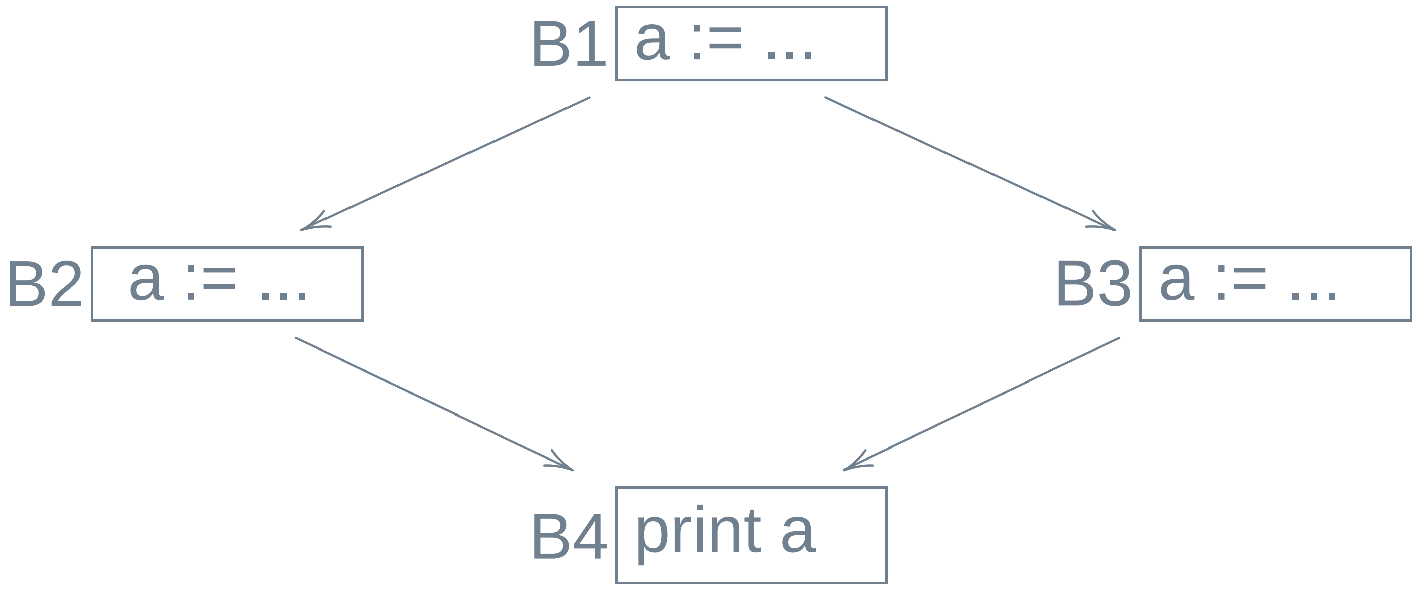


Figure 1: A Dead Store Variable

Dead Code Elimination

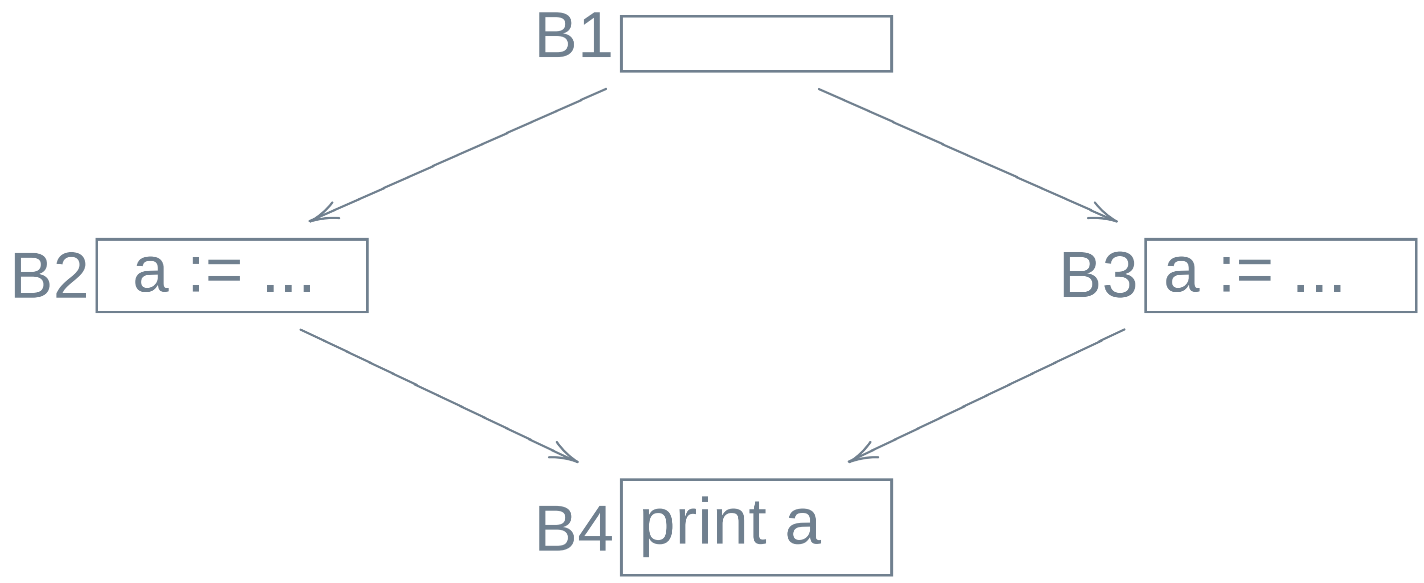


Figure 2: A Dead Store Eliminated

Partially Dead Code

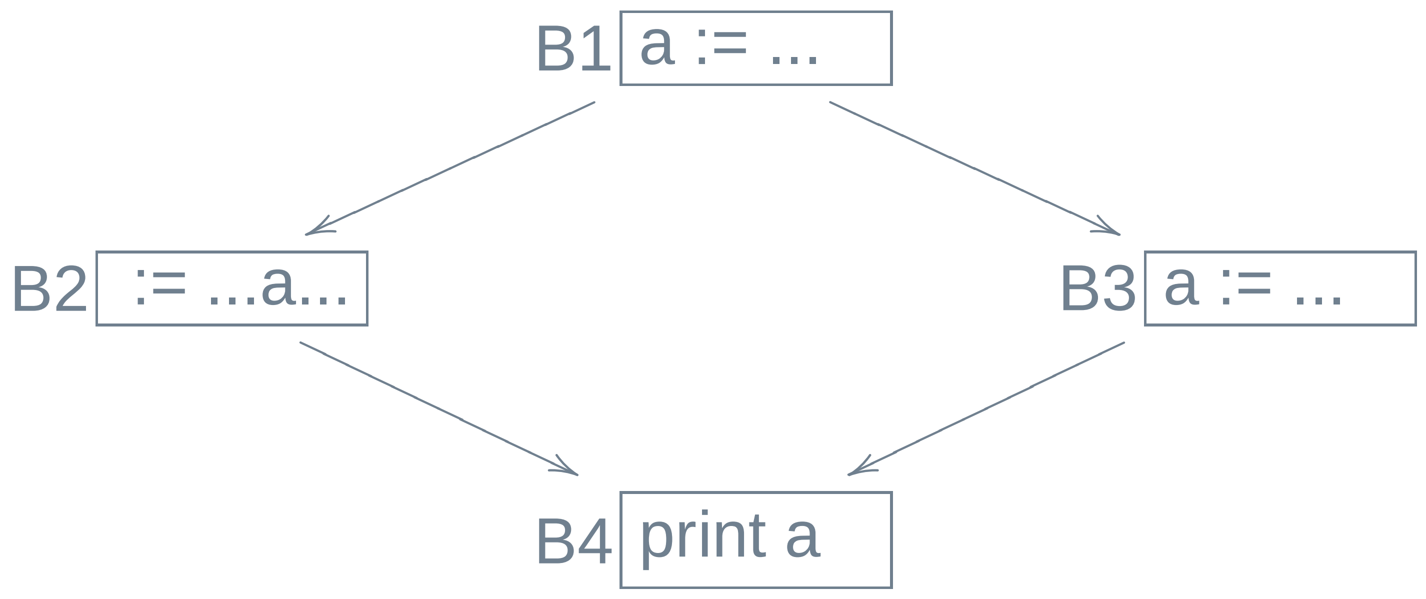


Figure 3: A Partially Live Store Variable

Step 1: Insert Stores At Use-Points

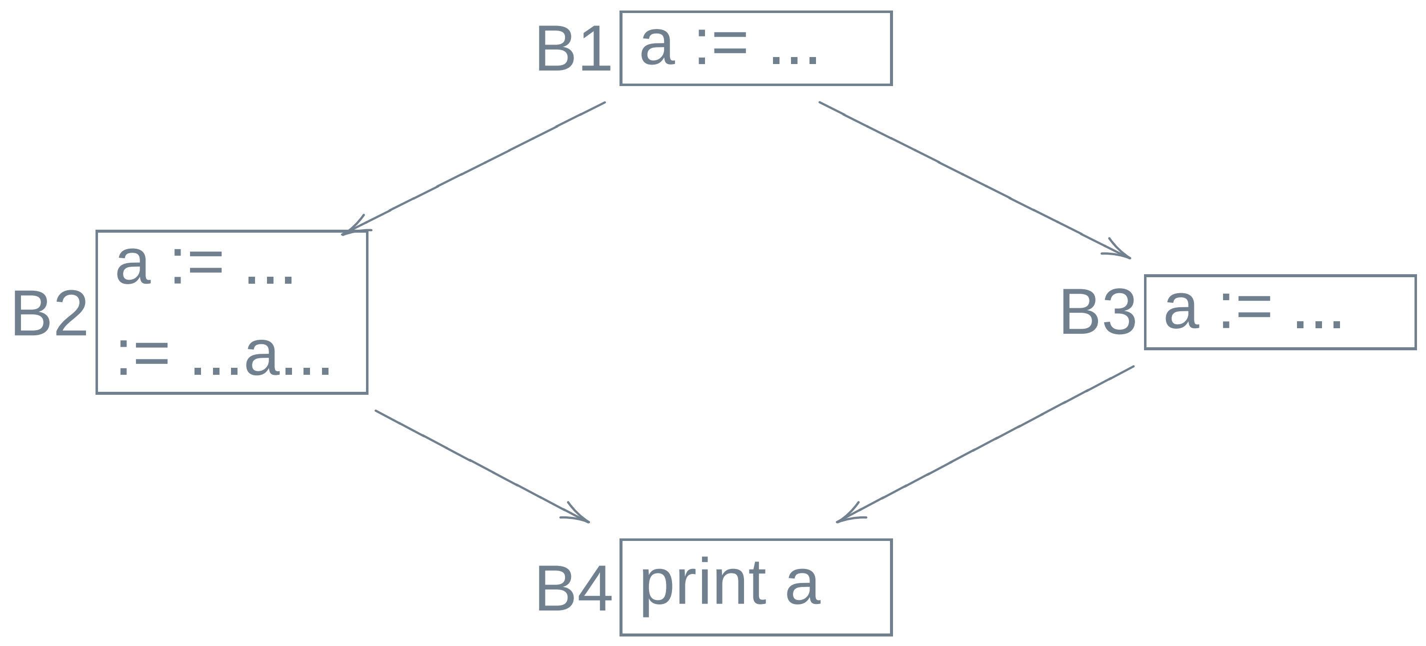


Figure 4: “Promotion” to a Completely Dead Store

Step 2: Eliminate Dead Stores

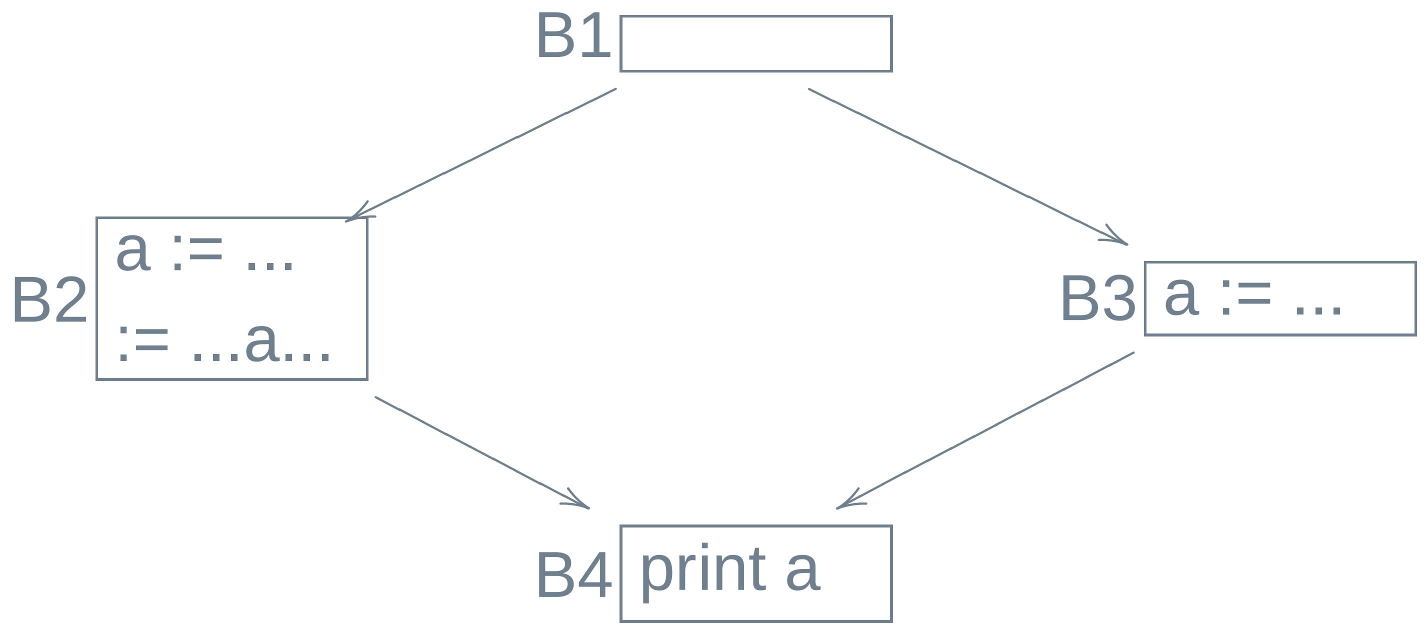


Figure 5: Partially Dead Store Eliminated

Speculative PDCE Algorithm

- Choose a threshold execution frequency, Θ.

Speculative PDCE Algorithm

- Choose a threshold execution frequency, Θ.
- Divide CFG into hot and cold regions.

Speculative PDCE Algorithm

- Choose a threshold execution frequency, Θ.
- Divide CFG into hot and cold regions.
- Split **egress** edges - Edges from the hot region to the cold region.

Speculative PDCE Algorithm

- Choose a threshold execution frequency, Θ.
- Divide CFG into hot and cold regions.
- Split **egress** edges - Edges from the hot region to the cold region.
- Insert stores on those edges.

Speculative PDCE Algorithm

- Choose a threshold execution frequency, Θ.
- Divide CFG into hot and cold regions.
- Split **egress** edges - Edges from the hot region to the cold region.
- Insert stores on those edges.
- Perform elementary dead code elimination.

Note

We present this algorithm in Static Single Assignment form.

A PDCE problem in SSA

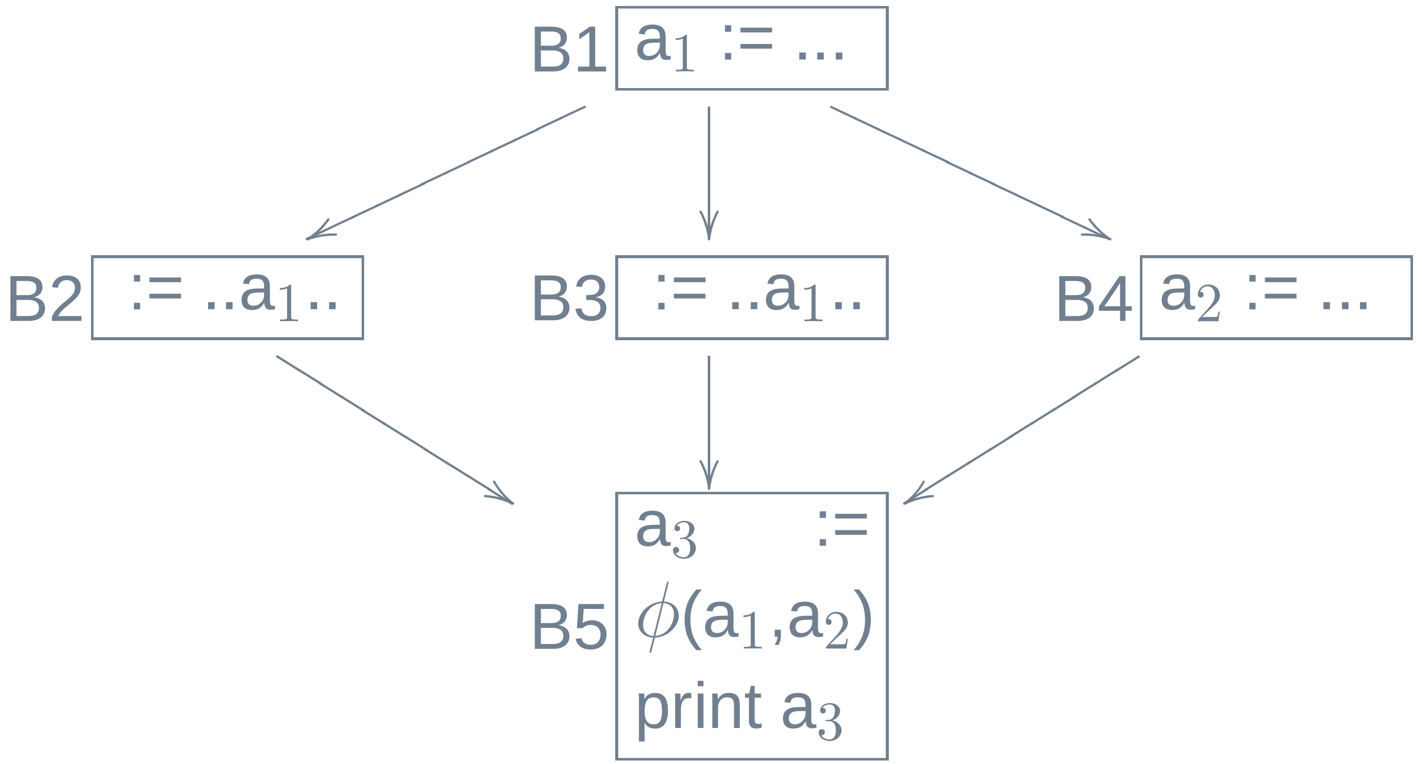


Figure 6: Reformulation of Figure 1 in SSA

... with execution frequencies

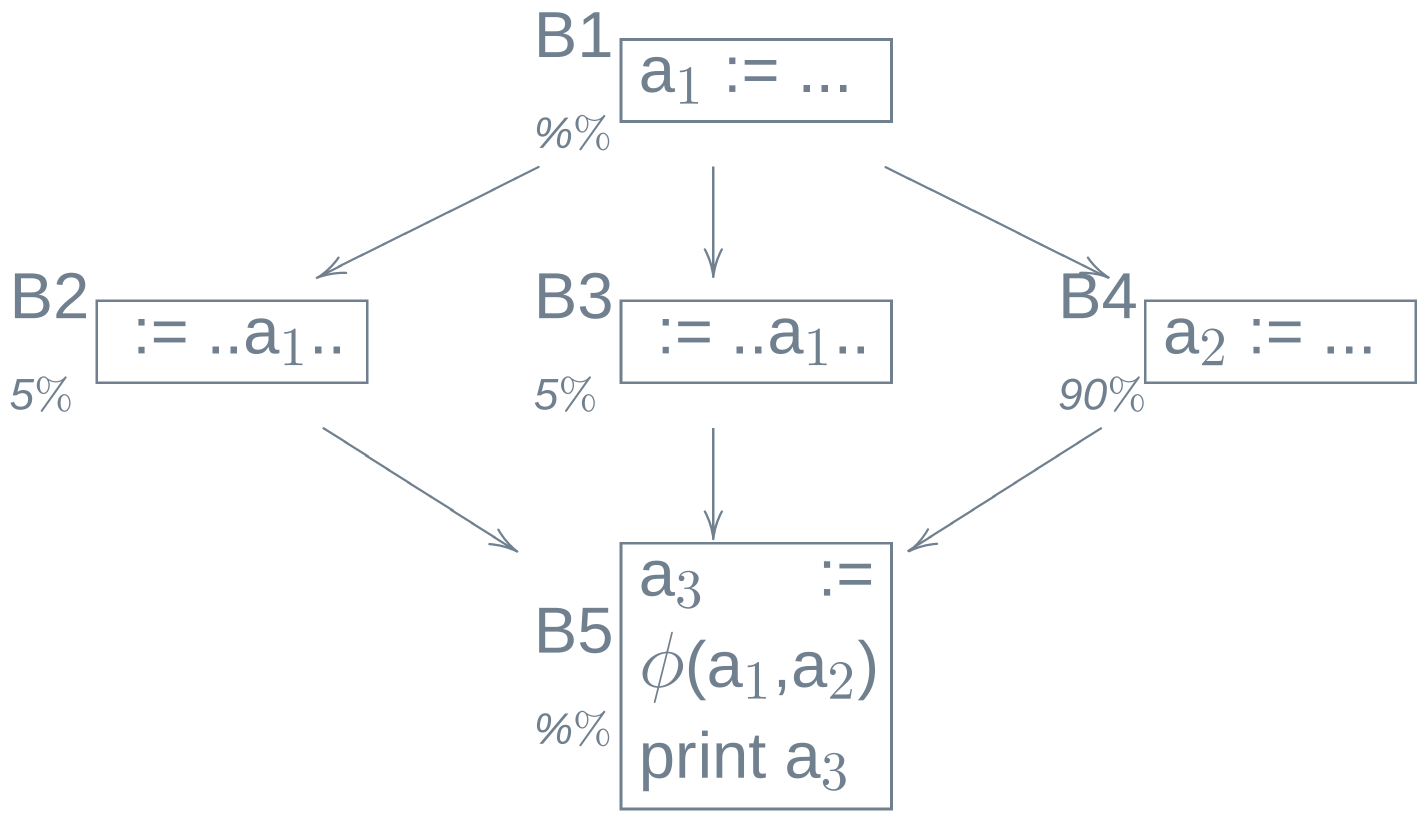


Figure 7: Reformulation of Figure 1 in SSA

Outline of Algorithm

- Partition graph to determine, hot region, cold region and egress edges.

Outline of Algorithm

- Partition graph to determine, hot region, cold region and egress edges.
- Discover Partially Live Stores (PLSs) in the hot region.

Outline of Algorithm

- Partition graph to determine, hot region, cold region and egress edges.
- Discover Partially Live Stores (PLSs) in the hot region.
- **Introduce:** For each discovered PLS (v = x):

Outline of Algorithm

- Partition graph to determine, hot region, cold region and egress edges.
- Discover Partially Live Stores (PLSs) in the hot region.
- **Introduce:** For each discovered PLS (v = x):
    - Place

Outline of Algorithm

- Partition graph to determine, hot region, cold region and egress edges.
- Discover Partially Live Stores (PLSs) in the hot region.
- **Introduce:** For each discovered PLS (v = x):
    - Place
    - Rename

Outline of Algorithm

- Partition graph to determine, hot region, cold region and egress edges.
- Discover Partially Live Stores (PLSs) in the hot region.
- **Introduce:** For each discovered PLS (v = x):
  - Place
  - Rename
  - Integrate

Outline of Algorithm

- Partition graph to determine, hot region, cold region and egress edges.
- Discover Partially Live Stores (PLSs) in the hot region.
- **Introduce:** For each discovered PLS (v = x):
  - Place
  - Rename
  - Integrate
- **Eliminate:** Perform elementary dead code elimination.

Placement

- Determine the subset of egress edge on which to place RHS.

Placement

- Determine the subset of egress edge on which to place RHS.
- For each such egress edge,

Placement

- Determine the subset of egress edge on which to place RHS.
- For each such egress edge,
  - Create an Alternative Store Variable (ASV), a.

Placement

- Determine the subset of egress edge on which to place RHS.
- For each such egress edge,
  - Create an Alternative Store Variable (ASV), a.
  - Insert store a=x on the edge.

Placement in Action...

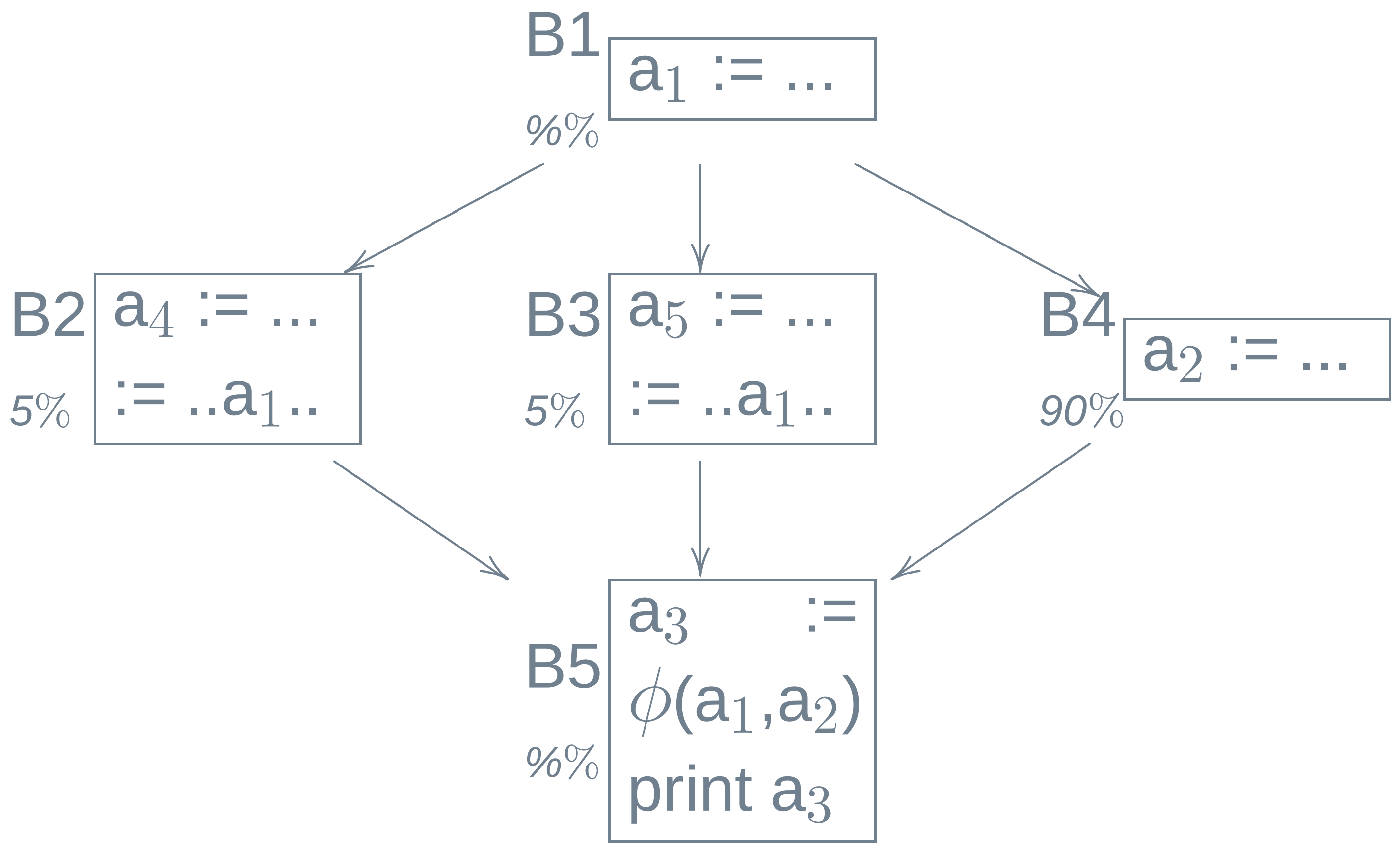


Figure 8: Placement of Stores to Alternate Store Variables

Rename

- Embed all newly placed stores into the dominator tree of the control flow graph.

Rename

- Embed all newly placed stores into the dominator tree of the control flow graph.
- Decorate dominator tree so that each node (basic block) is associated with closest store above it. This associates an ASV with each node

Rename

- Embed all newly placed stores into the dominator tree of the control flow graph.
- Decorate dominator tree so that each node (basic block) is associated with closest store above it. This associates an ASV with each node
- For each block, replace all uses of the PSLV with the block’s ASV.

Renaming in Action...

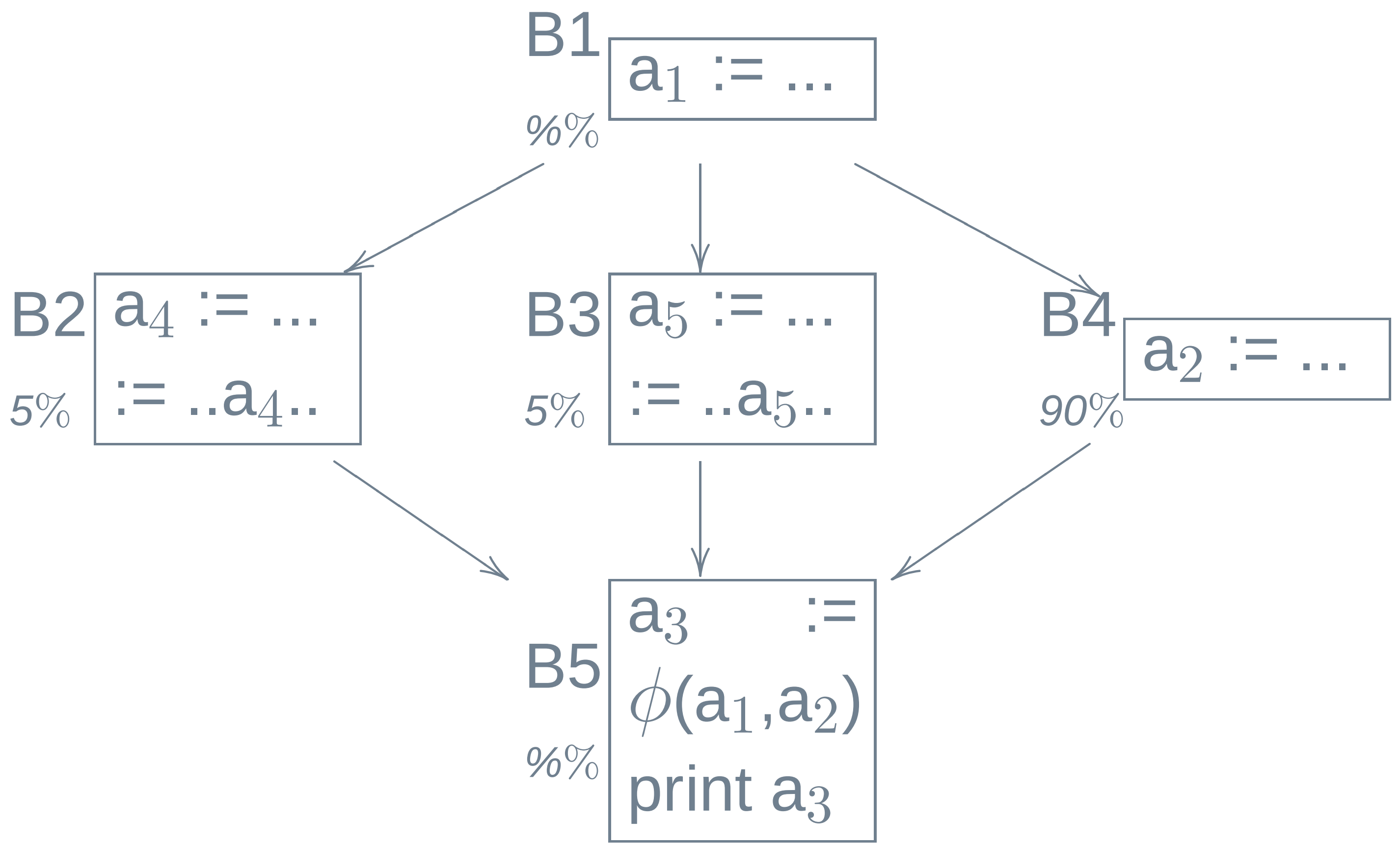


Figure 9: Renaming of uses of PLSV

Integrate

- Compute the subset of the Live ASVs.

Integrate

- Compute the subset of the Live ASVs.
- Add the LASV’s to the ϕ-functions where the original PSLV occurs.

Integrate

- Compute the subset of the Live ASVs.
- Add the LASV's to the ϕ-functions where the original PSLV occurs.
  - The ϕ-functions occur in the dominance *frontiers* and so would not have been modified by the renaming step.

Integration in Action...

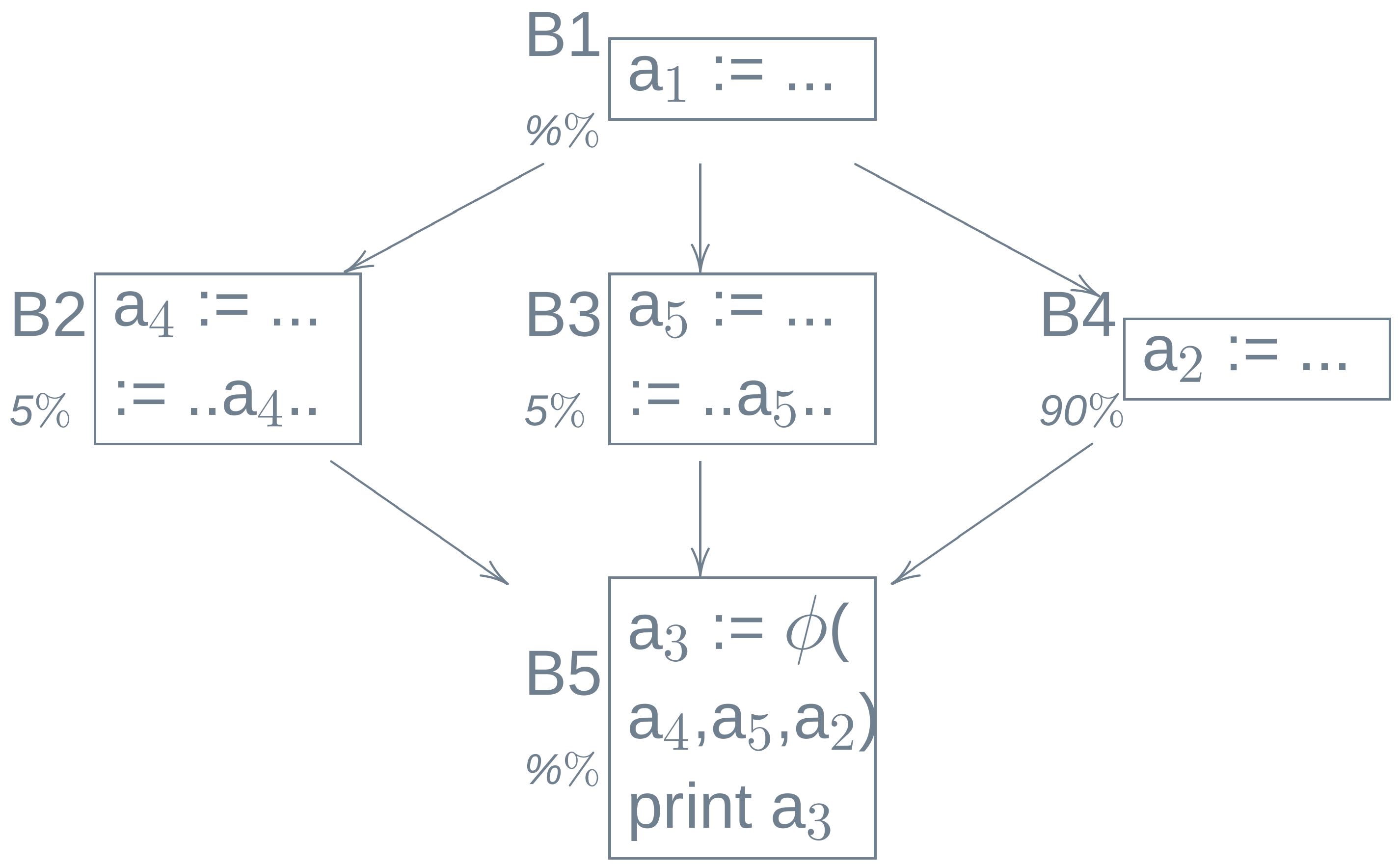


Figure 10: Renaming of uses of PLSV

Trivial DCE gives...

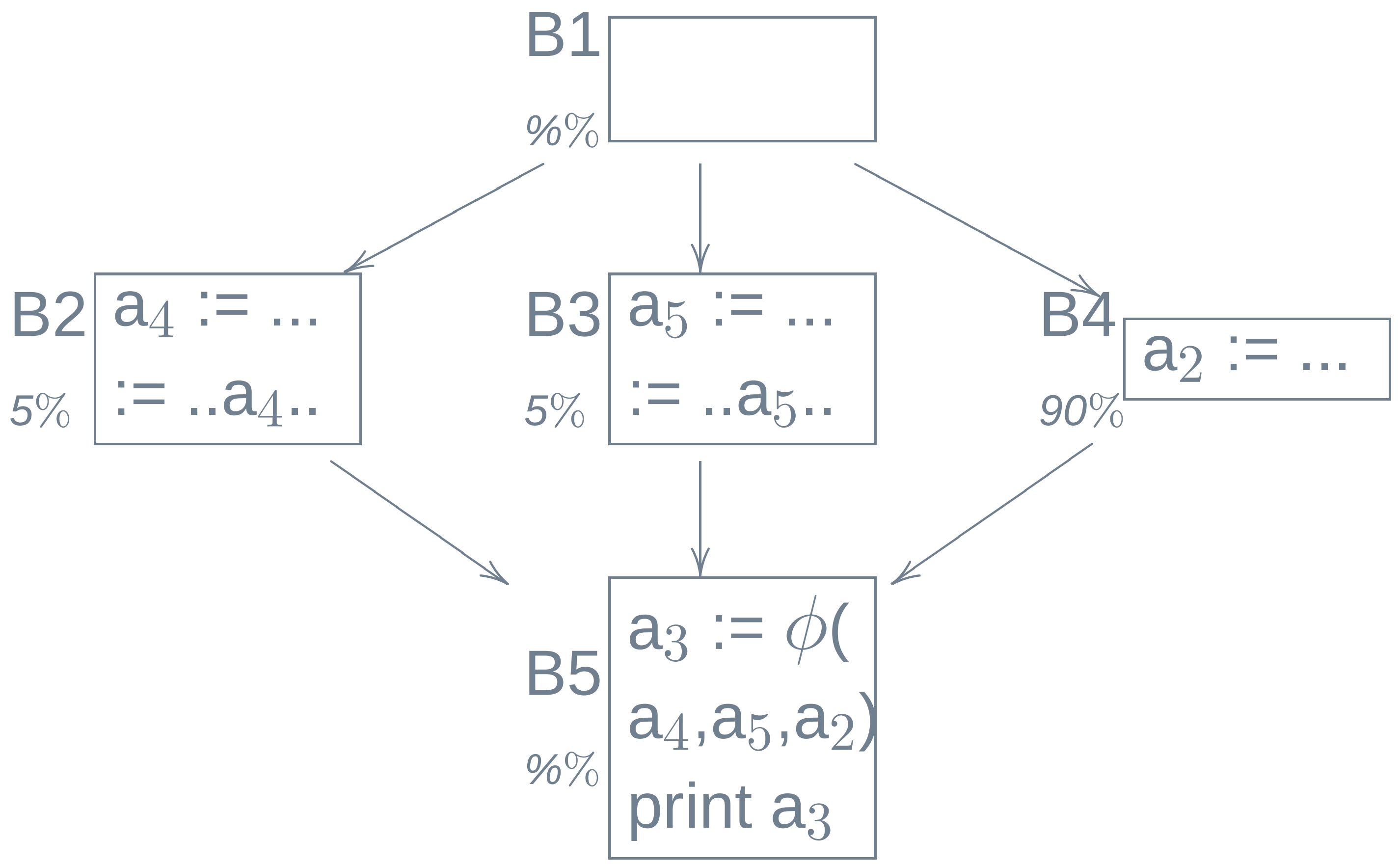


Figure 11: Renaming of uses of PLSV

Why SSA ?

- Dominance Based Rematerialization
- Preserving Checks

Dominance Based Rematerialization

- Consider a store into a PLSV (v = a+b).

Dominance Based Rematerialization

- Consider a store into a PLSV (v = a+b).
- We may attempt to insert the RHS (a+b) on an egress edge.

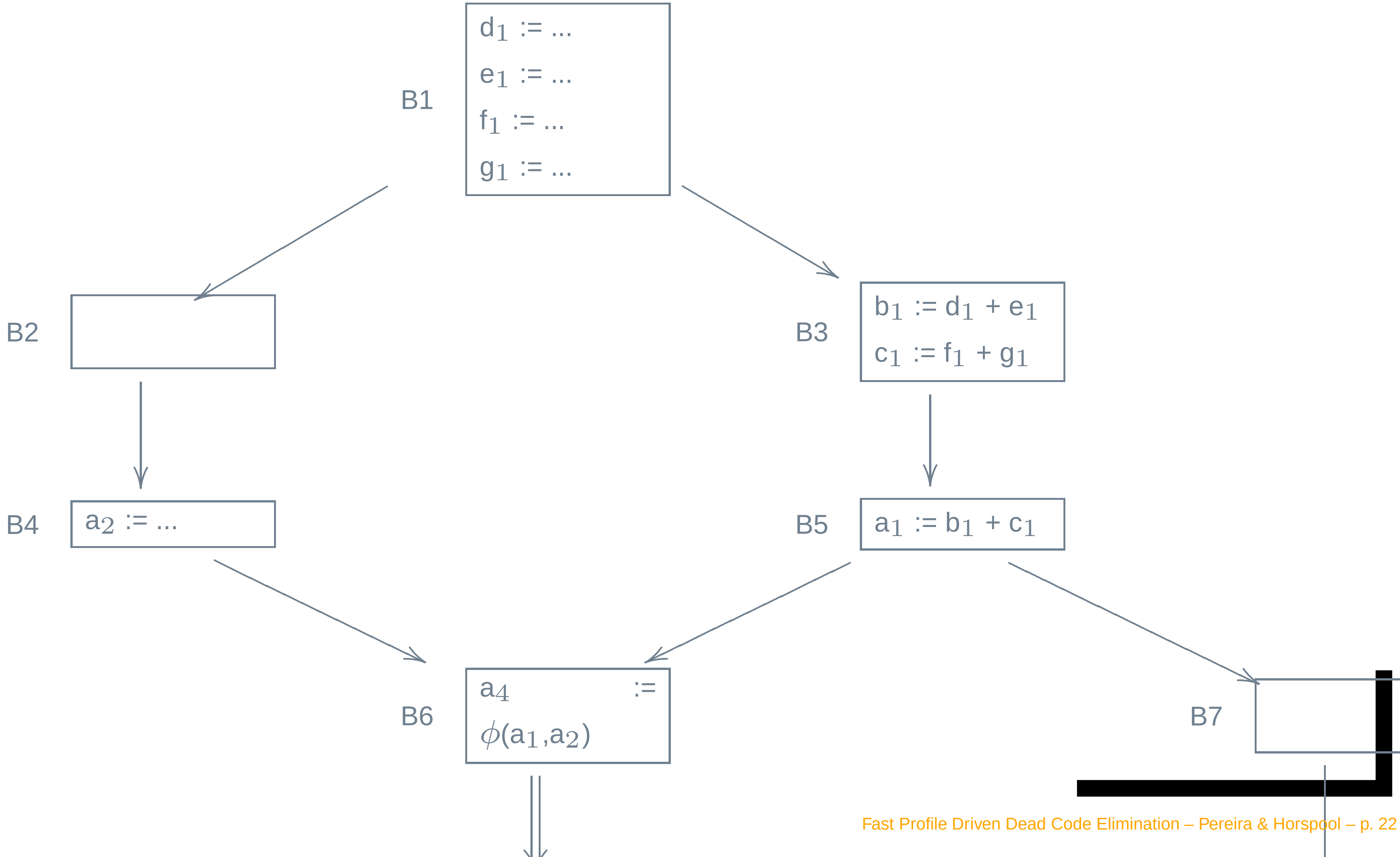
Dominance Based Rematerialization

- Consider a store into a PLSV (v = a+b).
- We may attempt to insert the RHS (a+b) on an egress edge.
- But the store may not dominate the egress edge

Dominance Based Rematerialization

- Consider a store into a PLSV (v = a+b).
- We may attempt to insert the RHS (a+b) on an egress edge.
- But the store may not dominate the egress edge
- But if the definition point of a and b dominate the edge we can recompute a and b where we need them.

Example: Rematerialization



Example: The Dominator Tree

$$\begin{tikzcd}
 & & B1 \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
B2 \arrow[d] & B3 \arrow[d] & B6 & B8 & B10 \\
B4 & B5 \arrow[d] & & & \\
 & B7 \arrow[d] & & & \\
 & B9 & & &
\end{tikzcd}$$

Figure 13: Dominator Tree for Example CFG

Check Preservation

... is preservation of dominance relationships

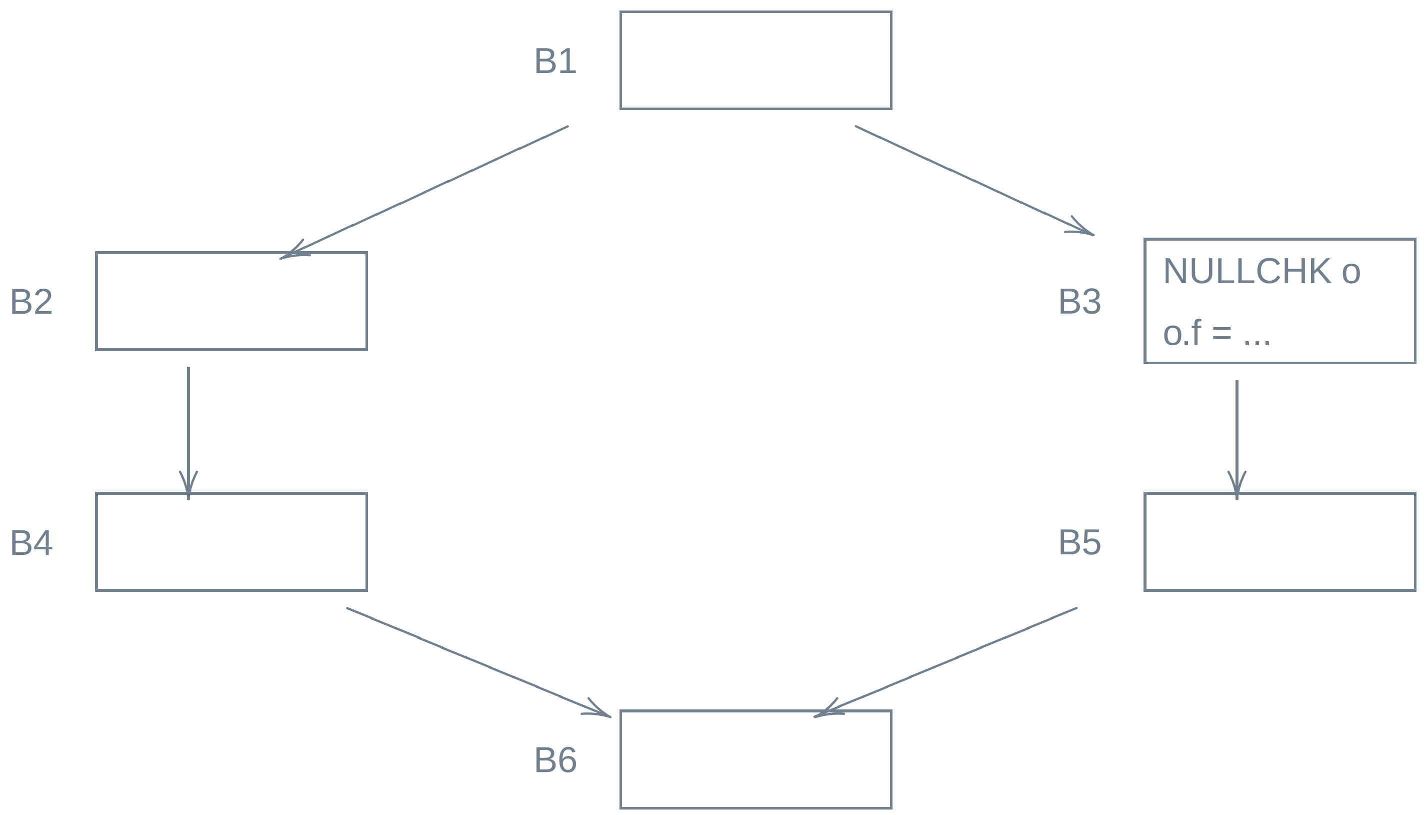


Figure 14: Store Motion Constrained

A Wrong Movement

... violates dominance relationships

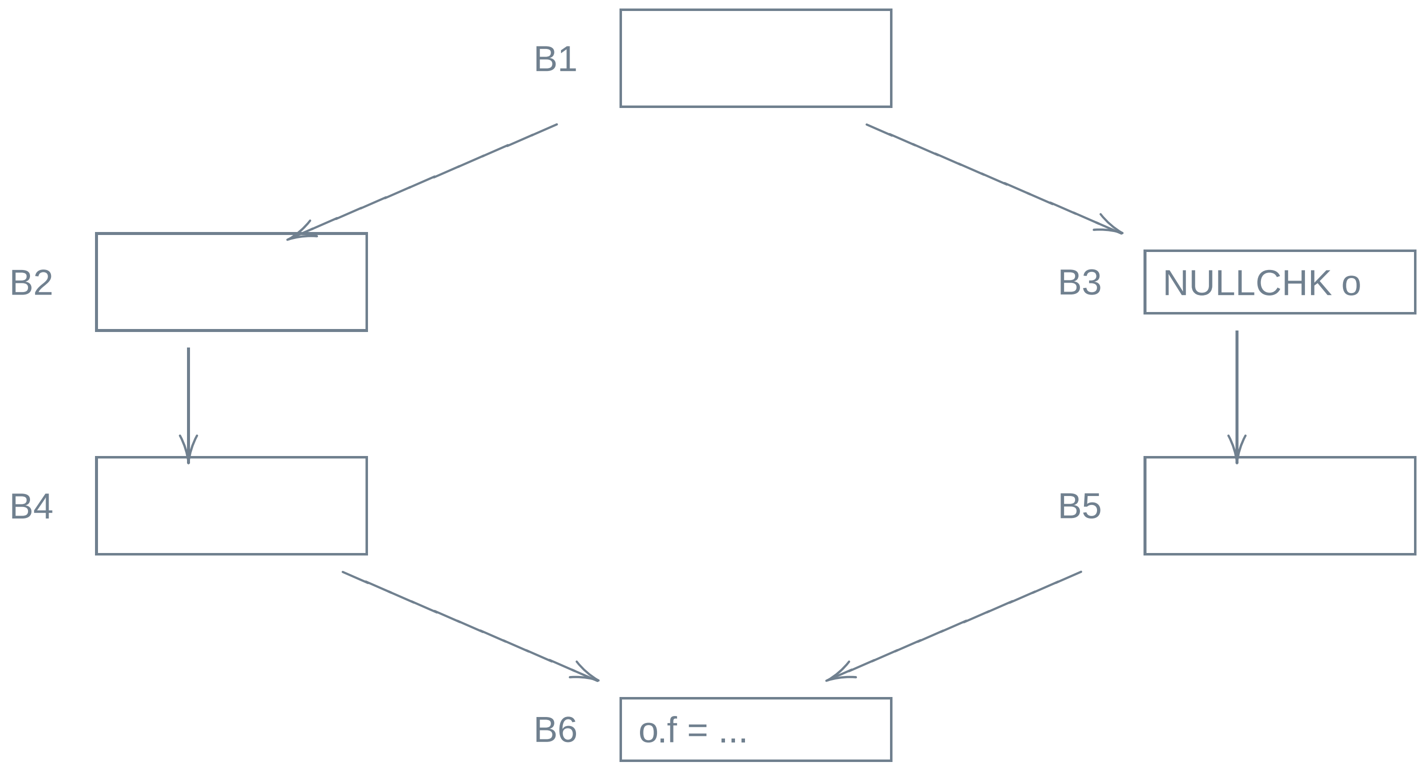


Figure 15: PLS No Longer Safe

Conclusions

- First PDCE algorithm to take profile into consideration

Conclusions

- First PDCE algorithm to take profile into consideration
- SSA formulation removed needs for complicated analysis.

Conclusions

- First PDCE algorithm to take profile into consideration
- SSA formulation removed needs for complicated analysis.
  - In fact, Knoop et al. have very complicated analysis to take interaction between expressions into consideration.

Conclusions

- First PDCE algorithm to take profile into consideration
- SSA formulation removed needs for complicated analysis.
    - In fact, Knoop et al. have very complicated analysis to take interaction between expressions into consideration.
- SSA ties in well with Java's checked instructions.

Conclusions

- First PDCE algorithm to take profile into consideration
- SSA formulation removed needs for complicated analysis.
  - In fact, Knoop et al. have very complicated analysis to take interaction between expressions into consideration.
- SSA ties in well with Java's checked instructions.
  - In fact, it provides a framework to think about them.

Conclusions

- First PDCE algorithm to take profile into consideration
- SSA formulation removed needs for complicated analysis.
  - In fact, Knoop et al. have very complicated analysis to take interaction between expressions into consideration.
- SSA ties in well with Java’s checked instructions.
  - In fact, it provides a framework to think about them.
- We are currently working on benchmarks.

Future Work

- SSA has hidden costs:

We need to determine how these costs offset the code motions we have done.

Future Work

- SSA has hidden costs:
  - Code Space

We need to determine how these costs offset the code motions we have done.

Future Work

- SSA has hidden costs:
  - Code Space
    - Array-SSA for Java is implemented by checking a timestamp vector.

We need to determine how these costs offset the code motions we have done.

Future Work

- SSA has hidden costs:
  - Code Space
    - Array-SSA for Java is implemented by checking a timestamp vector.
  - Data Space

We need to determine how these costs offset the code motions we have done.

Future Work

- SSA has hidden costs:
  - Code Space
    - Array-SSA for Java is implemented by checking a timestamp vector.
  - Data Space
    - To use an expression, we need to hold its (previous) value in a temp. SSA superficially ignores this...

We need to determine how these costs offset the code motions we have done.

Future Work

- SSA has hidden costs:
    - Code Space
        - Array-SSA for Java is implemented by checking a timestamp vector.
    - Data Space
        - To use an expression, we need to hold its (previous) value in a temp. SSA superficially ignores this...
        - But ultimately, it has to put it in.

We need to determine how these costs offset the code motions we have done.

Future Work

- SSA has hidden costs:
  - Code Space
    - Array-SSA for Java is implemented by checking a timestamp vector.
  - Data Space
    - To use an expression, we need to hold its (previous) value in a temp. SSA superficially ignores this...
    - But ultimately, it has to put it in.

We need to determine how these costs offset the code motions we have done.

Question Period