

# Eliminating memory references

Joshua Dunfield and Alina Oprea

Computer Science Department  
Carnegie Mellon University

# Measuring load reuse

- Undecidable

# Measuring load reuse

- Undecidable
- But, can *simulate* program execution

# Simulation

- For each memory instruction, keep an *access history*

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )
- Limit history depth by a constant  $h$

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )
- Limit history depth by a constant  $h$
- $h \geq 2$  allows us to identify reuse in the loop

```
for (i=0; i<N-2; i++) {  
    r := A[i]  
    A[i+2] := r  
}
```

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )
- Limit history depth by a constant  $h$
- $h \geq 2$  allows us to identify reuse in the loop

```
for (i=0; i<N-2; i++) {  
    r := A[i]      A[0]  
    A[i+2] := r  
}
```

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )
- Limit history depth by a constant  $h$
- $h \geq 2$  allows us to identify reuse in the loop

```
for (i=0; i<N-2; i++) {  
    r := A[i]    A[0]  
    A[i+2] := r  A[2]  
}
```

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )
- Limit history depth by a constant  $h$
- $h \geq 2$  allows us to identify reuse in the loop

```
for (i=0; i<N-2; i++) {  
    r := A[i]      A[0]  A[1]  
    A[i+2] := r   A[2]  
}
```

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )
- Limit history depth by a constant  $h$
- $h \geq 2$  allows us to identify reuse in the loop

```
for (i=0; i<N-2; i++) {  
    r := A[i]      A[0]  A[1]  
    A[i+2] := r    A[2]  A[3]  
}
```

# Simulation

- For each memory instruction, keep an *access history*
- A dynamic load from  $p$  is redundant if we can remember a previous access of  $p$  (with no intervening store to  $p$ )
- Limit history depth by a constant  $h$
- $h \geq 2$  allows us to identify reuse in the loop

```
for (i=0; i<N-2; i++) {  
    r := A[i]      A[0]  A[1]  A[2]  
    A[i+2] := r   A[2]  A[3]  
}
```

# Simulation: Issues

- What history depth  $h$  do we need?

# Simulation: Issues

- What history depth  $h$  do we need?
- ...in tests (SPECint95),  $h = 4$  only slightly better than  $h = 1$

# Simulation: Issues

- What history depth  $h$  do we need?
- ...in tests (SPECint95),  $h = 4$  only slightly better than  $h = 1$
- Variation among program inputs

# Simulation: Issues

- What history depth  $h$  do we need?
- ...in tests (SPECint95),  $h = 4$  only slightly better than  $h = 1$
- Variation among program inputs
- ...in tests, typically within 18% (“largely input independent”) across “several” inputs

# Simulation: Issues

- What history depth  $h$  do we need?
- ... in tests (SPECint95),  $h = 4$  only slightly better than  $h = 1$
- Variation among program inputs
- ... in tests, typically within 18% (“largely input independent”) across “several” inputs
- ... could take *intersection* of reuses over all inputs

# Simulation: Issues

- What history depth  $h$  do we need?
- ...in tests (SPECint95),  $h = 4$  only slightly better than  $h = 1$
- Variation among program inputs
- ...in tests, typically within 18% (“largely input independent”) across “several” inputs
- ...could take *intersection* of reuses over all inputs
- Only count reuse where generator and load are from the same procedure

# Simulation: Issues

- What history depth  $h$  do we need?
- ... in tests (SPECint95),  $h = 4$  only slightly better than  $h = 1$
- Variation among program inputs
- ... in tests, typically within 18% (“largely input independent”) across “several” inputs
- ... could take *intersection* of reuses over all inputs
- Only count reuse where generator and load are from the same procedure
- But, doesn't exclude reuse *across* calls to same procedure

# Simulation: Issues

- What history depth  $h$  do we need?
- ... in tests (SPECint95),  $h = 4$  only slightly better than  $h = 1$
- Variation among program inputs
- ... in tests, typically within 18% (“largely input independent”) across “several” inputs
- ... could take *intersection* of reuses over all inputs
- Only count reuse where generator and load are from the same procedure
- But, doesn't exclude reuse *across* calls to same procedure
- ... clear history at procedure exit?

# Simulation

- The simulator also returns an *edge profile* for use in the estimator

# Estimator

## Input:

- A data-flow solution from the load-reuse analysis
- An edge profile from the simulator

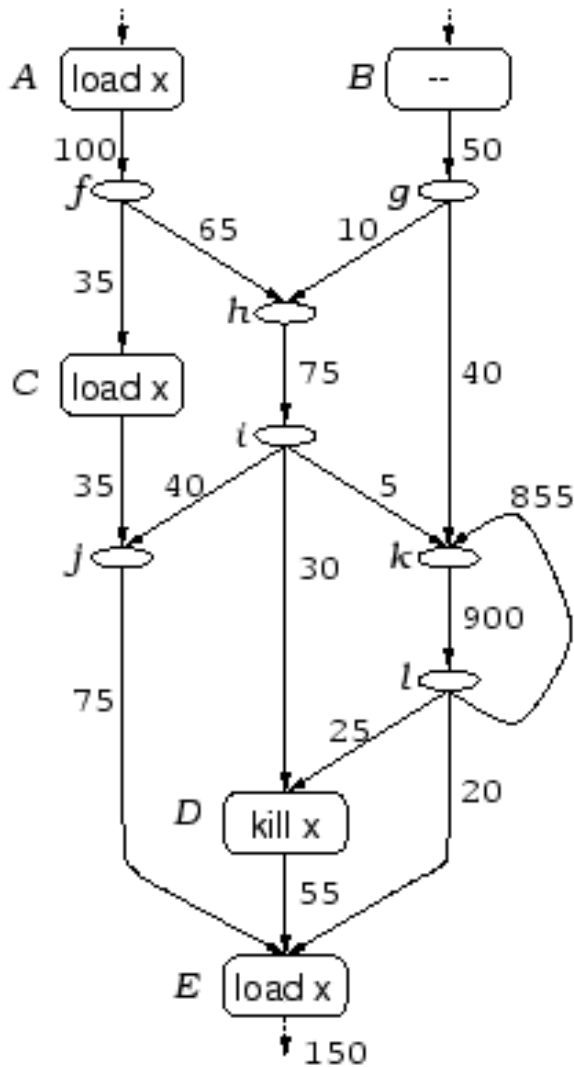
## Output:

- A *weighted* data-flow solution

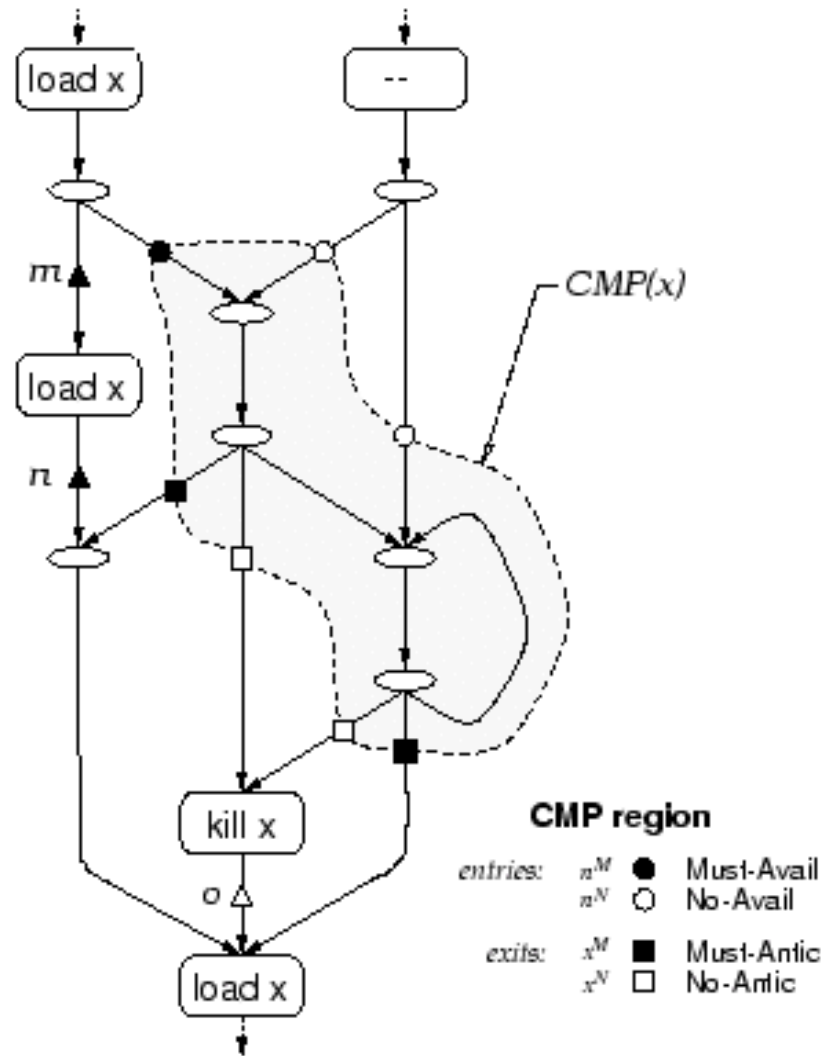
## Purpose:

- For comparison to the the actual reuse measured by the simulator
- (For use in profile-driven code transformations)

# Estimator: Method



(a) The source program annotated with an edge profile.



(b) The **CMP** region for the reuse on memory location  $x$ .

# Estimator: Accuracy

- Average error 5% – 15%

# Results

- The analysis detects about 80% of the reuse that could be eliminated by PRE (according to the estimator)

# Questions

- If you're simulating the program anyway (to find actual reuse), why not use an execution trace instead of an approximation (edge profile)?
- Analysis assumes infinite registers. Could we do better?
- How much would path profiles improve estimator accuracy?