Scalar Optimisation Part 2

Michael O'Boyle

January 2014

Course Structure

- L1 Introduction and Recap

- **4-5 lectures on classical optimisation**
  - 2 lectures on scalar optimisation
  - Last lecture on redundant expressions
  - Today look at dataflow framework and SSA

- 4-5 lectures on high level approaches

- 4-5 lectures on adaptive compilation

Dataflow analysis for redundant expressions: calculate available

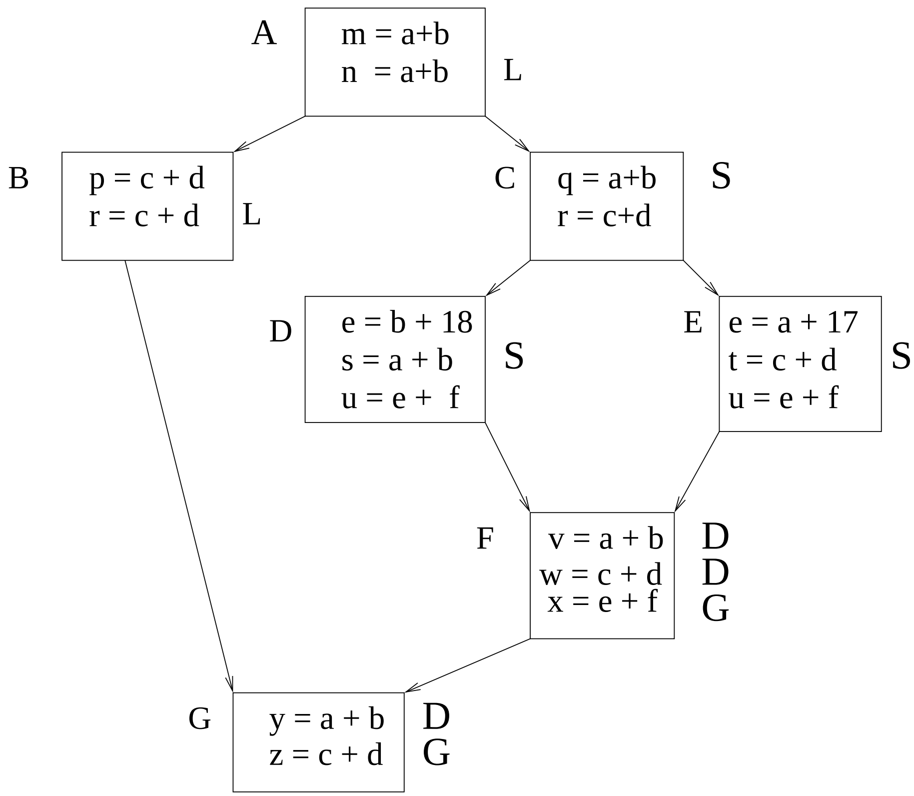
$DEExpr(b)$ - subexpressions not overwritten in this block b (local)

$NOTKILLED(b)$ - subexpressions that are not killed (local)

$AVAIL(b) = \bigcap_{p \in pred(b)} (DEExpr(p) \cup (AVAIL(p) \cap NOTKILLED(p)))$

- $DEExpr(b)$ and $NOTKILLED(b)$ can be calculated locally for each basic block b
- Initialise $AVAIL(b) = \emptyset$
- Find for each block in turn calculate $AVAIL(b)$ based on predecessors
- Keep repeating the procedure till results stabilise.

AVAIL() set calculation



Node	A	B	C	D	E	F	G
pred	-	A	A	C	C	D,E	B,F
DEExpr	a+b	c+d	a+b c+d	b+18 a+b e+f	a+17 c+d e+f	a+b c+d e+f	a+b c+d
Kill				e+f	e+f		

Calculate Avail(b) for each Basic Block b starting at block A

$AVAIL(A) = \emptyset$

$AVAIL(B) = (DEExpr(A) \cup (AVAIL(A) \cap NOTKILLED(A)))$
$= \{a + b\} \cup (\emptyset \cap U) = \{a + b\}$

$AVAIL(G) = (DEExpr(B) \cup (AVAIL(B) \cap NOTKILLED(B)))$
$\bigcap(DEExpr(F) \cup (AVAIL(F) \cap NOTKILLED(F)))$

Find available expressions

Post order

Node	A	B	C	D	E	F	G
Avail1	$\emptyset$	a+b	a+b	a+b,c+d	a+b,c+d	e+f	c+d
Avail2						a+b,c+d,e+f	a+b,c+d

Reverse Post order: Finds fixed point on first iteration

Node	A	B	C	D	E	F	G
Avail1	$\emptyset$	a+b	a+b	a+b,c+d	a+b,c+d	a+b,c+d,e+f	a+b,c+d

Traversal order affects number of iterations to solve equations.

Will solution always terminate?

How many iterations? What class of problems?

Another example: Dataflow analysis for live variables

- A variable v is live at a point p if there is a path from p to a use of v along which v is not redefined.
- Useful to eliminate stores of variables no longer needed - useless store elimination
- Useful for detecting uninitialised variables
- Essential for global register allocation
- Determines whether a variable MAY be read after this BB and is therefore a candidate to be put in a register

Equations for live vars

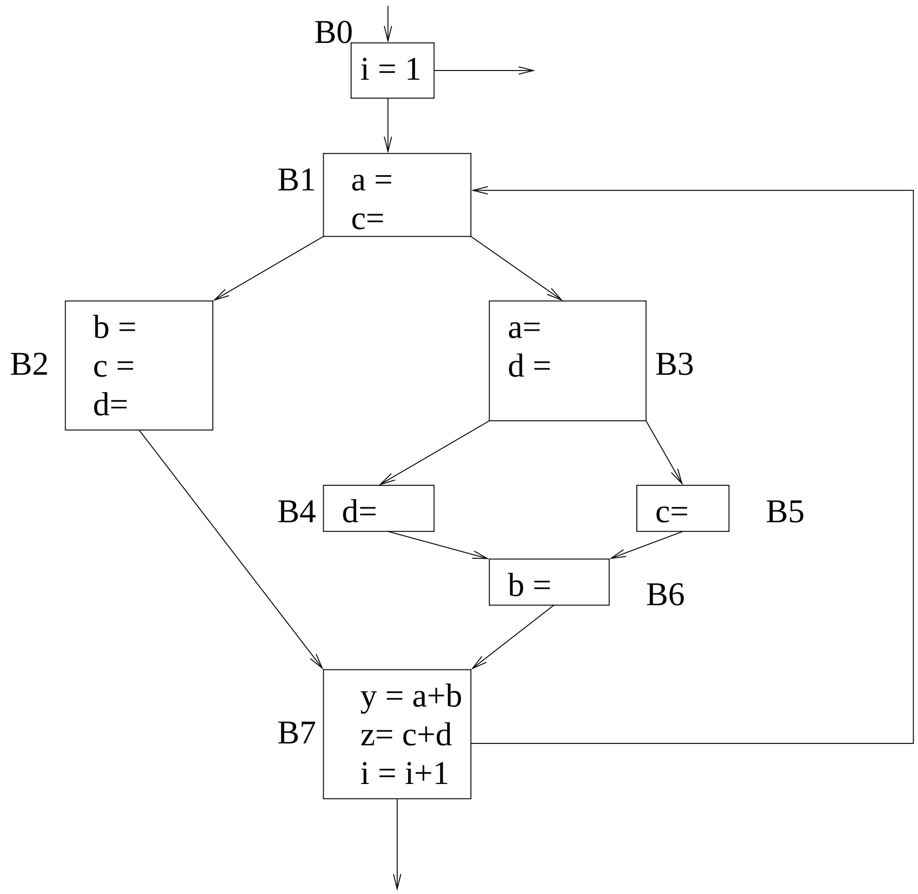
$LiveOut(b) = \bigcup_{p \in succ(b)} (UEVar(p) \cup (LiveOut(p) \cap NotKilledVar(p)))$

$UEVar(p)$ upwardly exposed variables used in p before redefinition

$NotKilledVar(p)$ var not defined in this block p

- Similar to AVAIL
- Depends on successors not predecessors backward vs forward
- AVAIL is an all paths problem ($\cap$) LiveOut any path ($\cup$)
- Can also be solved using iterative algorithm. (How long/terminate?)

Example of LiveOut



Solution:

	B0	B1	B2	B3	B4	B5	B6	B7
UEVar	-	-	-	-	-	-	-	a,b,c,d,i
NVarKill	a,b,c,d,y,z	b,d,i,y,z	a,i,y,z	b,c,i,y,z	a,b,c,i,y,z	a,b,d,i,y,z	a,c,d,i,y,z	a,b,c,d

Reverse Post order

Iter	B0	B1	B2	B3	B4	B5	B6	B7
0	-	-	-	-	-	-	-	-
1	-	-	a,b,c,d,i	-	-	-	a,b,c,d,i	-
2	-	a,i	a,b,c,d,i	-	a,c,d,i	a,c,d,i	a,b,c,d,i	i
3	i	a,i	a,b,c,d,i	a,c,d,i	a,c,d,i	a,c,d,i	a,b,c,d,i	i
4	i	a,c,i	a,b,c,d,i	a,c,d,i	a,c,d,i	a,c,d,i	a,b,c,d,i	i
5	i	a,c,i	a,b,c,d,i	a,c,d,i	a,c,d,i	a,c,d,i	a,b,c,d,i	i

5 iterations to fixed point. Is this the quickest solution?

Solution 2

Post order

Iter	B0	B1	B2	B3	B4	B5	B6	B7
0	-	-	-	-	-	-	-	-
1	i	a,c,i	a,b,c,d,i	a,c,d,i	a,c,d,i	a,c,d,i	a,b,c,d,i	-
2	i	a,c,i	a,b,c,d,i	a,c,d,i	a,c,d,i	a,c,d,i	a,b,c,d,i	i
3	i	a,c,i	a,b,c,d,i	a,c,d,i	a,c,d,i	a,c,d,i	a,b,c,d,i	i

- What is the best order?

- Question: why does all this work?

Semi-lattice

A set L and a meet operator $\wedge$ such that

$\forall a, b, c \in L$

1. $a \wedge a = a$

2. $a \wedge b = b \wedge a$

3. $a \wedge (b \wedge c) = (a \wedge b) \wedge c$

$\wedge$ imposes an order $a \geq b \rightarrow a \wedge b = b$

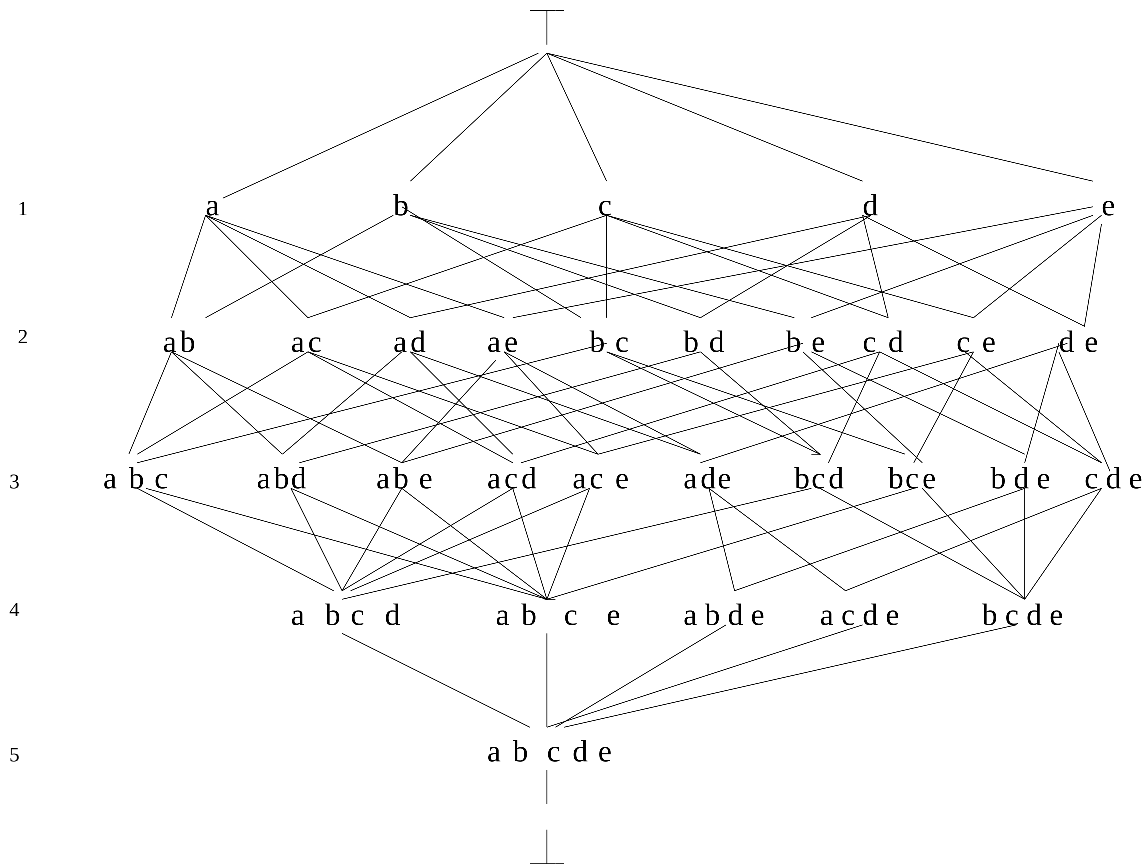
Contains a bottom element $\perp$, $\perp \wedge a = \perp, a \geq \perp$

Models an ordered finite set of facts

Semi-lattice

- Choose a semi-lattice to represent the facts
- Attach a meaning to each $a \in L$. Each a distinct set of facts
- For each node (basic block) n in the CFG, associate a function $f_n : L \mapsto L$.
- It models the behaviour of the code belonging to n
- Avail: Semilatice is $(2^E, \wedge)$,E the set of all expressions, $\wedge$ is $\cap$.

Example of LiveOut lattice



Round Robin algorithm

```
for i = 1 to N
  Avail(b[i]) = 0
change =true
while (change)
  change =false
  for i = 0 to N
    temp = intersect[i]  (Def(x) union (Avail x union Nkill(x)))
    if avail(b[i]) != temp
      change = true
       avail(b[i]) =temp
```

Standard algorithm to solve dataflow. There are faster ones.

Iterative data flow

- If f is monotone and the semi-lattice bounded then the round robin algorithm terminates and finds a least fixed point
- Given certain technical constraints on f, there is a unique fixed point and **order of evaluation does not matter**
- Pick an order that converges quickly
- A lot of theory about this. Given certain conditions then a round-robin post-order alg will finish in $d(G) + 3$ passes where d(G)is the loop connectedness
- Most dataflow fits this. Means runs in linear time. Muchnick for more details for more in depth explanation.

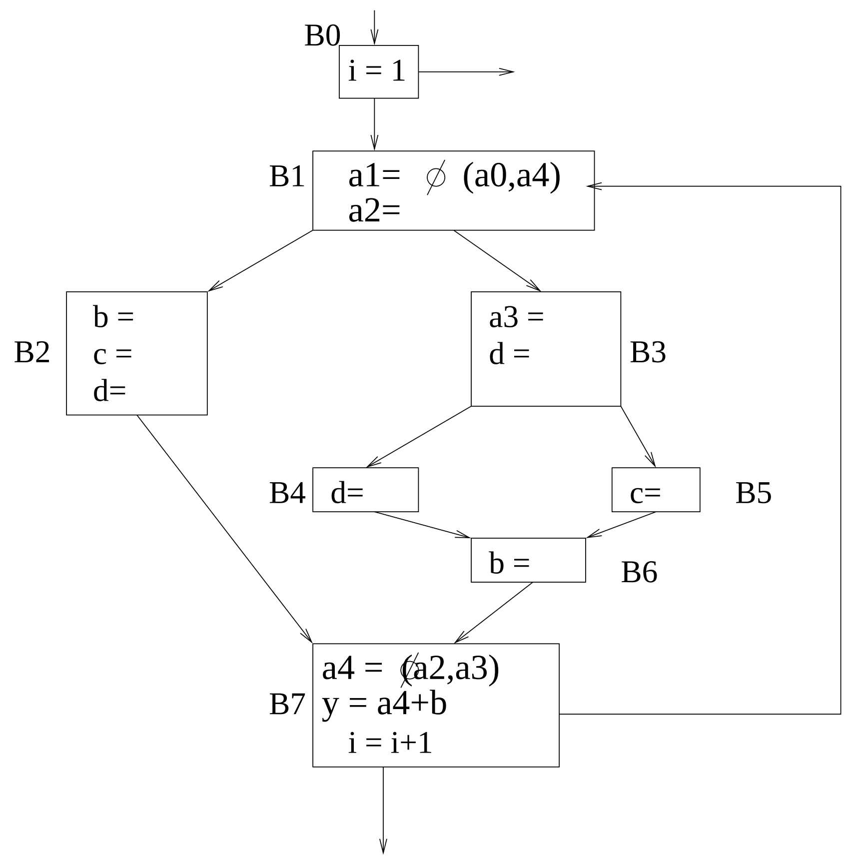
Other dataflow analysis

- Reaching definitions : Find all places where a variable was defined and not killed subsequently
- Very Busy Expressions: An expression is evaluated on all paths leaving a block - used for code hoisting
- Constant Propagation. Shows that a variable v has the same value at point p regardless of control-flow. Allows specialisation.
- Uses a very small lattice and terminates quickly. Easy to express using SSA form

SSA form

- Most advanced analysis needs to track def and uses of vars rather than basic block summary
- Variables can have multiple definitions and uses
- Need to keep track of which def flows to which use over all possible control-flow paths
- SSA gives a unique name to each definition
- Need ϕ nodes to handle merging of control-flow
- Can be constructed in $O(n)$ time. Increasingly standard form.

Example SSA



Algorithms using SSA

- Many dataflow algorithms are considerably simplified using SSA
- Value numbering. Each value has a unique name allowing value numbering on complex control-flow
- $Constants(n) = \bigwedge_{p \in pred(n)} F_p(Constants(p))$
- Small lattice $\top >$ {-maxint .. +maxint} $> \perp$
- Meet operator: $\top \wedge x = x, \perp \wedge x = \perp, c_i \wedge c_j = c_i if c_i = c_j else \perp$
- F_p depends on the operations in block. Optimistic algorithm

Algorithms using SSA: Constant propagation

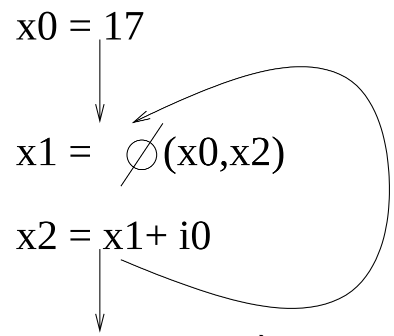
Model F_p

x $=$ y if Constants(p) $= \{(x, c_1), (y, c_2), ..\}$ then
Constants(p) $=$ Constants(p) - $(x, c_1) \cup (x, c_2)$

- eg update old value of x (c1) with the new value in y (c2)

x $=$ y op z if Constants(p) $= \{(x, c_1), (y, c_2), (z, c_3)..\}$ then
Constants(p) $=$ Constants(p) - $(x, c_1) \cup (x, c_2 op c_3)$

- eg update old value of x (c1) with the new value after (c2 op c3)



	x0	x1	x2 (when i0 $=$ 0)
0	17	$\top$	$\top$
1	17	$17 \wedge \top = 17$	$17 - \{17\} \cup \{17 + 0\} = 17$
2	17	$17 \wedge 17 = 17$	$17 - \{17\} \cup \{17 + 0\} = 17$

	x0	x1	x2 (when i0 $=$ 1)
0	17	$\top$	$\top$
1	17	$17 \wedge \top = 17$	$17 - \{17\} \cup \{17 + 1\} = 18$
2	17	$17 \wedge 18 = \perp$	$17 - \{17\} \cup \{\perp + 1\} = \perp$

Limits and Extensions

- Dataflow assumes that all paths in the CFG are taken hence conservative approximations
- Guarded SSA attempts to overcome this by having additional meet nodes γ, η and μ to carry conditional information around
- Arrays considered monolithic objects A[1] $=$ .., $=$A[2] considered a def-use
- Array based SSA models access patterns - can be generalised using presburger formula
- Inter-procedural challenging. Pointers destroy analysis! Large research effort in points-to analysis.

Summary

- Levels of optimisations
- Examined dataflow as a generic optimisation framework
- Round robin algorithm and lattices
- Using SSA as a framework for optimisation
- Limits of dataflow -other techniques?
- Next lecture code generation.