Scalar Optimisation Part 1

Michael O'Boyle

January, 2014

Course Structure

- L1 Introduction and Recap
- **4/5 lectures on classical optimisation**
  - 2 lectures on scalar optimisation
  - Today example optimisations
  - Next lecture dataflow framework and SSA
- 5 lectures on high level approaches
- 4-5 lectures on adaptive compilation

Overview

- Machine dependent vs independent optimisations
- Redundant elimination example
  - Local value numbering
  - Super value numbering
  - Dominator value numbering
- Alternative general approach
  - Global Redundancy Elimination
  - Based on iterative dataflow analysis
- Other dataflow analysis: Live variable analysis

Optimisation Classification

- Machine independent vs dependent - not always a clear distinction. Main trends in architecture increased memory latency and exploitation of ILP are machine dependent

- Machine independent applicable to all. Eliminate redundant work, accesses. Use less expensive operations where possible

- Optimisation can be performed at source, IR, assembler, machine code level.

- Concentrate on machine independent scalar optimisation - IR level.

- Optimisation = analysis + transformation . Form depends on IR - impact on complexity.

Redundant expression elimination

An expression `x + y` is redundant if already evaluated and not redefined

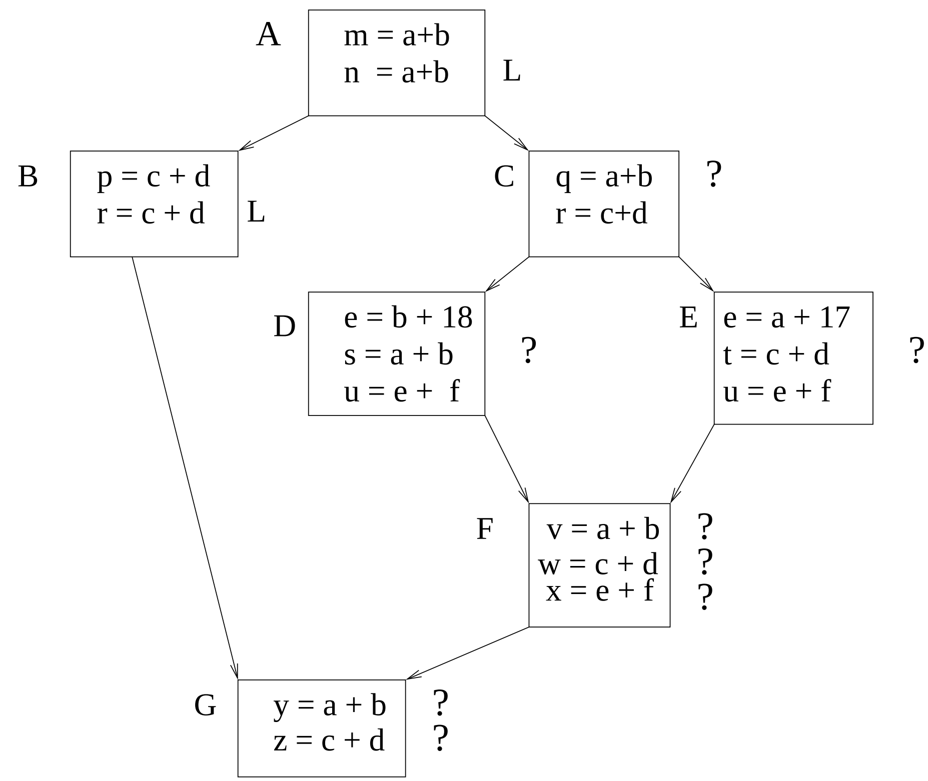
Value numbering: Associate numbers with operators/operands and hash lookup in table Hash (+,x,y) return value number

If value number already there replace with reference to variable

$a^3 = x^1 + y^2$	$a^3 = x^1 + y^2$	$a_0^3 = x_0^1 + y_0^2$	$a_0^3 = x_0^1 + y_0^2$
$b^4 = x^1 + y^2$	$b^4 = a^3$	$b_0^4 = x_0^1 + y_0^2$	$b_0^4 = a_0^3$
$a^3 = 17$	$a^3 = 17$	$a_1^3 = 17$	$a_1^3 = 17^4$
$c^5 = x^1 + y^2$	$c^5 = a^3$!!	$c_0^5 = x_0^1 + y_0^2$	$c_0^5 = a_0^3$

Can be extended to handle larger scope based on dominators. Fails in presence of general control-flow

Example: CFG rep of program. Basic blocks + control-flow.



LVN removes some but not all of redundant expressions: L vs ?

Super Local Value numbering SVN

Basic blocks(BB) have just one entry and exit.

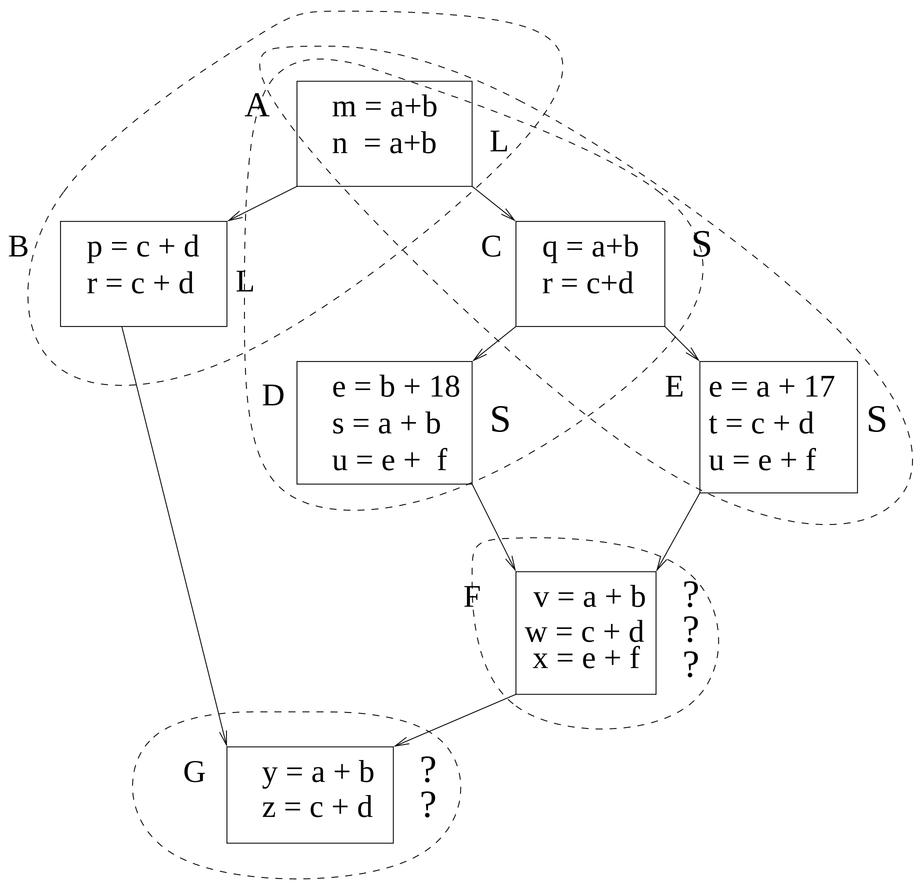
- Extended BB: (EBB) A tree of BBs $\{B_1, \ldots, B_n\}$ where B_1 may have multiple predecessors.
- All others have a single unique predecessor but possibly multiple exits.
- This tree is only entered at the root.

In our example 3 EBBs (A,B,C,D,E), (F), (G)

SVN considers each path within an EBB as single block

So (A,B), (A,C,D), (A,C,E) are considered paths for LVN

Example: Extended Basic Blocks .



Dominator Value numbering DVN

SVN based on EBBs fail when there are join paths in the graph.

- Use concept of dominators. Basic idea if reaching paths to a node share common ancestor nodes, then these can be used for redundancy elimination

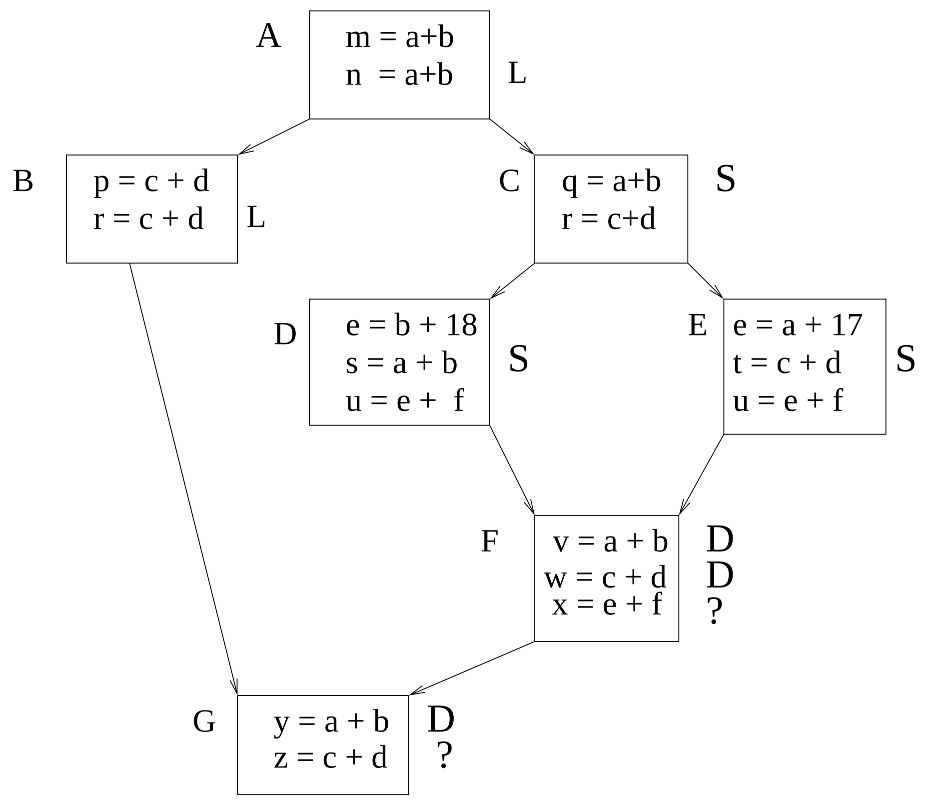
- A node X strictly dominates Y ($X >> Y$) if $X \neq Y$ and if X appears on every path from the graph entry to Y

Node	A	B	C	D	E	F	G
DOM	-	A	A	A,C	A,C	A,C	A
IDOM	-	A	A	C	C	C	A

IDOM- immediate dominator - forms a dominator tree.

So if expression appears in F but defined in A,C then redundant

Example: Dominator value numbering.



What about the remaining two ?s in G and F?

Dataflow analysis

- A formal program analysis that has a wide range of application.
- Described property of a program at a particular point in set based recurrence equations
- Assumes a control-flow graph(CFG) consisting of nodes (basic blocks) and edges: control-flow
- Determines property at a point in the program as a function of local information and approximation of global information
- Approx solutions will converge to exact solution in finite number of iterations for finite lattices - more detail next lecture

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 $AVAIL(b) = \emptyset$
- For each block in turn calculate $AVAIL(b)$ based on predecessors
- Keep repeating the procedure till results stabilise.

Find available expressions part 1

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

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

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

Find available expressions part 2

D and E are the same

$AVAIL(D) = (DEExpr(C) \cup (AVAIL(C) \cap NOTKILLED(C)))$
$= \{a+b, c+d\} \cup (\{\mathbf{a+b}\} \cap U) = \{a+b, c+d\}$

$AVAIL(E) = (DEExpr(C) \cup (AVAIL(C) \cap NOTKILLED(C)))$
$= \{a+b, c+d\} \cup (\{\mathbf{a+b}\} \cap U) = \{a+b, c+d\}$

F is a join point: 2 predecessors

$AVAIL(F) = (DEExpr(D) \cup (AVAIL(D) \cap NOTKILLED(D)))$
$\bigcap(DEExpr(E) \cup (AVAIL(E) \cap NOTKILLED(E)) =$
$\{b+18, a+b, e+f\} \cup (\{\mathbf{a+b}, \mathbf{c+d}\} \cap U - \{e+f\})$
$\bigcap\{a+17, c+d, e+f\} \cup (\{\mathbf{a+b}, \mathbf{c+d}\} \cap U - \{e+f\})$
$= \{a+b, c+d, e+f\}$

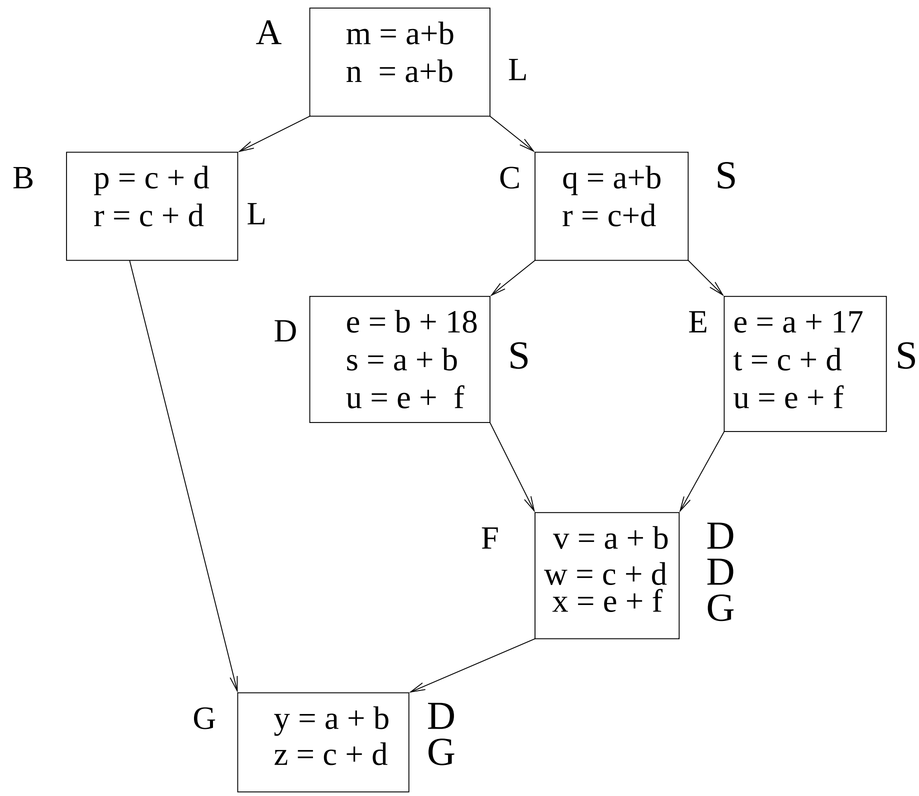
Find available expressions part 3

G another join point

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

Calculate this one yourselves

Example: Global redundancy elim using AVAIL()



Summary

- Levels of optimisations
- Redundant expression elimination
- LVN, SVN, DVN
- Introduced dataflow as a generic optimisation framework
- Iterative solution to equations
- Next time: More detailed examination of dataflow and SSA