Regional Optimization:
Superlocal Value Numbering

Comp 412
Material for the Second Exam

Lecture
• “Parsing, V” through “Superlocal Value Numbering”
• Excluding the material on local register allocation and lab 2 (15 to 18)

Textbook
• Sections 3.4 & 3.5
  – We will not ask you to build LR(1) tables on the exam
• Chapters 5, 6, & 7
• Sections 8.1, 8.2, 8.3, 8.4.1, 8.5.1

We will post a practice exam today or tomorrow.
for i ← 1 to n

1. get the value numbers $V_1$ for $L_i$ and $V_2$ for $R_i$
2. if $Op_i$ commutes and $V_1 > V_2$ then
   swap $V_1$ and $V_2$
3. construct a hash key $<V_1, Op_i, V_2>$
4. if the hash key is already present in the table then
   replace operation $i$ with a copy into $T_i$, and mark $T_i$ with the VN
5. if $V_1$ and $V_2$ are both marked as constant then
   evaluate $L_i \, Op_i \, R_i$, assign it to $T_i$, enter it, and mark $T_i$ as a constant
6. if $L_i \, Op_i \, R_i$ matches an identity then
   replace operation $i$ with the simplified version or a copy assignment, and
   enter it
7. Else enter $T_i$ and the hash key into the table

“enter it” means to create new table entries for $T_i$ & the hash key and assign them a new value number.
A Multi-Block Example

Control-flow graph (CFG)
- Nodes for basic blocks
- Edges for branches
- Basis for much of program analysis & transformation

Control-flow graph $G = (N, E)$
- $N = \{A, B, C, D, E, F, G\}$
- $E = \{(A,B), (A,C), (B,G), (C,D), (C,E), (D,F), (E,F), (F,E)\}$
- $|N| = 7$, $|E| = 8$

See § 5.3.4 in EaC2e for CFG construction algorithm
A Multi-Block Example

Local Value Numbering (LVN)
- 1 block at a time
- Strong local results
- No inter-block effects

LVN finds redundant ops in red
A Multi-Block Example

Local Value Numbering (LVN)
- 1 block at a time
- Strong local results
- No inter-block effects

LVN finds redundant ops in red
LVN misses redundant ops in blue
Beyond a Basic Block: Extended Basic Blocks

An Extended Basic Block (EBB)

- Set of blocks $B_1, B_2, ..., B_n$
- $B_1$ has $>1$ predecessor
- All other $B_i$ have 1 pred. & that pred. is in the EBB
Extended Basic Blocks

An Extended Basic Block (EBB)
- Set of blocks $B_1, B_2, \ldots, B_n$
- $B_1$ has $\geq 1$ predecessor
- All other $B_i$ have 1 pred. & that pred. is in the EBB

Three EBBs in this CFG
1. \{A, B, C, D, E\}
Extended Basic Blocks

An Extended Basic Block (EBB)
- Set of blocks $B_1, B_2, \ldots, B_n$
- $B_1$ has $> 1$ predecessor
- All other $B_i$ have 1 pred. & that pred. is in the EBB

Three EBBs in this CFG
1. $\{ A, B, C, D, E \}$
2. $\{ F \}$
Extended Basic Blocks

An Extended Basic Block (EBB)

- Set of blocks $B_1, B_2, ..., B_n$
- $B_1$ has > 1 predecessor
- All other $B_i$ have 1 pred. & that pred. is in the EBB

Three EBBs in this CFG
1. \{ A, B, C, D, E \}
2. \{ F \}
3. \{ G \}
Extended Basic Blocks

An Extended Basic Block (EBB)
- Set of blocks $B_1, B_2, ..., B_n$
- $B_1$ has > 1 predecessor
- All other $B_i$ have 1 pred. & that pred. is in the EBB

Three EBBs in this CFG
1. $\{ A, B, C, D, E \}$
2. $\{ F \}$
3. $\{ G \}$  
   Degenerate or trivial EBBs

The image contains a graph with nodes labeled A to G and equations associated with each node. The equations are:

- A: $m \leftarrow a + b$, $n \leftarrow a + b$
- B: $p \leftarrow c + d$, $r \leftarrow c + d$
- C: $q \leftarrow a + b$, $r \leftarrow c + d$
- D: $e \leftarrow b + 18$, $s \leftarrow a + b$, $u \leftarrow e + f$
- E: $e \leftarrow a + 17$, $t \leftarrow c + d$, $u \leftarrow e + f$
- F: $v \leftarrow a + b$, $w \leftarrow c + d$, $x \leftarrow e + f$
- G: $y \leftarrow a + b$, $z \leftarrow c + d$
Extended Basic Blocks

An Extended Basic Block (EBB)

- Set of blocks $B_1, B_2, ..., B_n$
- $B_1$ has $> 1$ predecessor
- All other $B_i$ have 1 pred. & that pred. is in the EBB

This definition is important

When we return to scheduling in the last week of classes, we will look at scheduling on EBBs. Important to remember: EBB is the set, individual paths are just paths in an EBB
Conceptual View of the Strategy

Local optimization is strong relative to that done in larger scopes

• Compiler writers try to apply local optimization to larger domains.
  – Find a “trick” that lets the optimization pretend it has a single block
  – Domain where all operations execute in a known order

• In general, local optimization across a branch is manageable
  – Can use knowledge from A in B and C
  – Moving code between these blocks takes care

• In general, local optimization across a merge is harder
  – Must merge state from D and E for use in F
  – Need “largest common set of facts” for F

EBBs capture the branches and cut off the merges
Value Numbering Over Extended Basic Blocks

Superlocal VN (SVN)
- Apply LVN to each path in EBB
- Carry hash table forward, block to block

Apply LVN to each path in EBB
1. (A, B)
Value Numbering Over Extended Basic Blocks

Superlocal VN
- Apply LVN to each path in EBB
- Carry hash table forward, block to block

Apply LVN to each path in EBB
1. (A, B)
2. (A, C, D)
Value Numbering Over Extended Basic Blocks

Superlocal VN
- Apply LVN to each path in EBB
- Carry hash table forward, block to block

Apply LVN to each path in EBB
1. (A, B)
2. (A, C, D)
3. (A, C, E)
And, of course, to (F) & (G)
Superlocal Value Numbering

**Efficiency**

- Easy to implement if we are willing to process A three times & C twice
  - A, AB, A, AC, ACD, A, AC, ACE, F, G
- Could be faster if we reused the results from A & C
  - A, AB, AC, ACD, ACE, F, G
Superlocal Value Numbering

Efficiency

• Easy to implement if we are willing to process A three times & C twice
  – A, AB, A, AC, ACD, A, AC, ACE, F, G

• Could be faster if we reused the results from A & C
  – A, AB, AC, ACD, ACE, F, G

Worst Case

• Imagine SVN on a case statement
Names in Superlocal Value Numbering

What work must be repeated in a predecessor block?

- Value numbers are stored in a hash table
  - Keyed by name or \(<\text{op,vn,vn}>\) construct
- To avoid repeated work, **SVN** should roll back changes to the hash table
  - Rather than A, AB, A, AC we want to go from AB to AC without revisiting A

In the example, the definition of x in B changes the hash table entry for x

- After AB, **SVN** needs to roll x’s value number back to the value from A
  - Could run backward through B and “undo” each definition (*with bookkeeping*)
  - Or, we could reprocess A
- Better way is to rename so that each definition has a unique name
  → *We saw the same issue in LVN, in local register allocation, & in local scheduling.*
- We need a global name space with the right set of properties
Superlocal Value Numbering

**Efficiency**
• Easy to implement if we are willing to process A three times & C twice
  – A, AB, A, AC, ACD, A, AC, ACE, F, G
• Could be faster if we reused the results from A & C
  – A, AB, AC, ACD, ACE, F, G
  – Need an appropriate name space & a scoped hash table
    → *Alternative is to add lots of complex mechanism for kills & table management*

**Desired Name Space**
• Unique name for each definition
  – Name $\Leftrightarrow$ VN
• SSA name space is ideal

*Use of scoped table & SSA is due to Simpson* (see [53] in EaC2e)
Aside: SSA Name Space

Two principles

- Each name is defined by exactly one operation
- Each operand refers to exactly one definition

To reconcile these principles with real code

- Insert $\phi$-functions at merge points to reconcile name space
- Add subscripts to variable names for uniqueness

A $\phi$-function selects one of its operands, based on the control-flow path used to reach the block.

Chapter 9 in EaC2e discusses construction & destruction of SSA form
Superlocal Value Numbering

With the right names, SVN becomes

1. Identify EBBs

2. In depth-first order over an EBB, starting with the head of the EBB, $b_0$
   a. Apply LVN to $b_i$
   b. Invoke SVN on each of $b_i$’s EBB successors

   → When going from $b_i$ to its EBB successor $b_j$, extend the symbol table with a new scope for $b_j$, apply LVN to $b_j$, & process $b_j$’s EBB successors

   → When going from $b_j$ to its EBB predecessor $b_i$, discard the scope for $b_j$

It is that easy, with a scoped table & the right name space
SVN on the Example

LVN finds redundant ops in red
SVN finds redundant ops in blue
SVN on the Example

LVN finds redundant ops in red
SVN finds redundant ops in blue
Both miss redundancies in F & G
What facts hold true in F?

- Could compute some “least common truth” from facts in D and E
  - Looks hard
  - “u” has different names in D and E

- Could give up some facts to make “truth” easier
  - A & C are on every path to F (only A & C)
  - A is on every path to F

If we could find these blocks that dominate the paths to a merge, we could use them (DVNT)