Regional optimization

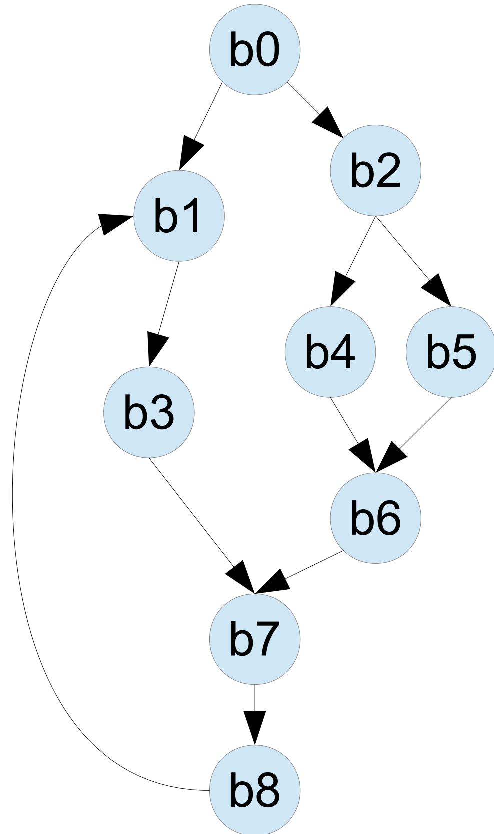
- regional optimizations look at some segment of code “around” one or more blocks (will refer to the regions as extended blocks)
- *in our data analysis section we'll look at ways to identify extended blocks, but each typically has a “gateway” statement (one you must pass through to enter the region), and expands to capture most local branching (e.g. for a loop)*
- some regional optimizations using another number value scheme, based on extensions to our basic blocks
- a variety of other optimizations apply specifically to loops, under the right circumstances

Superlocal value numbering

- like our LVN scheme for local optimization, expanded now to look at an extended basic block, EBB
- represent the EBB as a control flow graph, with basic blocks as the nodes and directed edges between them
- for SVN, basic blocks with multiple entry edges can only appear as the 'root' block in an EBB
- within an EBB, we can take each linear path of the EBB and apply local value numbering (and resulting optimizations) to it
- finds cross-block optimizations missed by single block LVN

Example: divide into EBBs and paths

- EBB1:
  b0, b2,
  b4, b5
- EBB2:
  b1, b3
- EBB3:
  b6
- EBB4:
  b7, b8



Possible LVN paths within EBBS
  path1: b0, b2, b4
  path2: b0, b2, b5
  path3: b1, b3
  path4: b6
  path5: b7, b8

each could be optimized as a simple linear block

*could improve compiler efficiency by recording b0+b2 optimizations while analyzing path 1, so they didn't have to be deveoped from scratch in path2

Loop optimization: unrolling

- replicate body multiple times, and run loop less often
- e.g. original loop:

```
for (i = 0; i < 1000; i++) { ...use i...; }
```

- gets replaced with

```
for (i = 1; i < 1000; i+=4) {
    ...use i...; ...use i...;
    ...use i...; ...use i...;
}
```

Pros/cons of loop unrolling

- cuts down number of times we test the loop condition and jump to start of loop (runs faster)
- the repetitive blocks of ...use i... are highly likely to be suitable for local optimization
- might also improve locality for cache hit ratio, and/or instruction cache (if one is in use)
- cost: adds extra lines of source code (larger executable)

unrolling while loop

```
while (x)  { body }
```

- can get unrolled as something like

```
while (x) {
    body
    if (!x) break;
    body
    if (!x) break;
    body
}
```

moving invariants outside loop

- suppose original loop has invariant test condition inside

```
while (x) { if (y) thing1; else thing2; }
```

- move the invariant test outside the loop

```
if (y) { while (x) thing1; } else { while (x) thing2; }
```

- improves execution speed, only tests y once
- might improve while loop optimization
- might improve cache hit ratio during while loop execution

loop splitting

- take large loop and break into sequence of smaller loops
- again, goal is to improve local optimizations, cache hits

```
for (i = 0; i < 1000; i++) { body }
```

- could be rewritten

```
for (i = 0; i < 250; i++) { body }
for (i = 250; i < 500; i++) { body }
for (i = 500; i < 750; i++) { body }
for (i = 750; i < 1000; i++) { body }
```