MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Frequent Path LICM

Due On: Feb 21, 2024

Loop Invariant Code Motion (LICM)

```
for (int i = 0; i< n; i++){
    x = y+z;
    a[i] = 6*i + x*x;
}
```

These values do not change within the body of the loop.

Loop Invariant Code Motion (LICM)

```
for (int i = 0; i< n; i++){
    x = y+z;
    a[i] = 6*i + x*x;
}
```

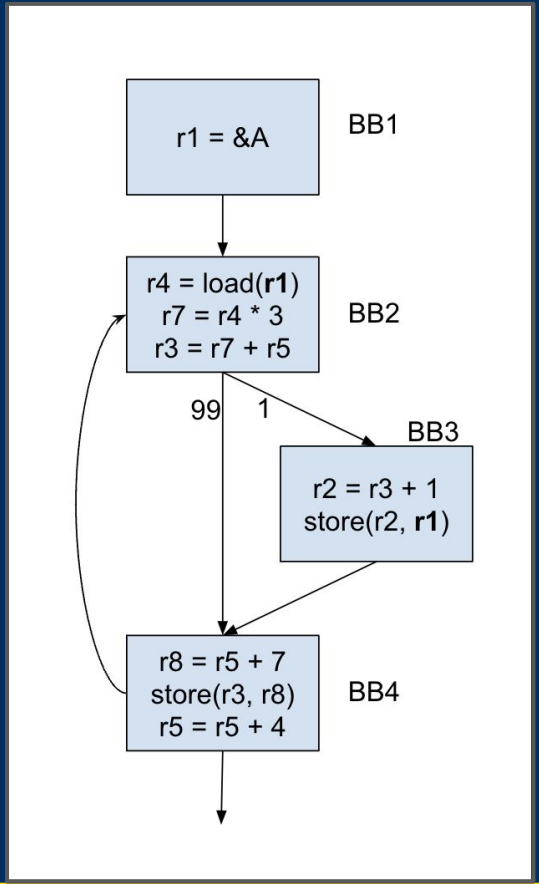
```
x = y+z;
int t1 = x*x;
for (int i = 0; i< n; i++){
    a[i] = 6*i + t1;
}
```

- Move operations whose source operands do not change within the loop to the loop preheader.
  - Execute them only 1x per invocation of the loop.
- LICM is already implemented in LLVM
  - /lib/Transforms/Scalar/LICM.cpp

Frequent Path LICM

- There is a store-load dependency.
- The load cannot be hoisted up because it is not invariant in the loop.

- But according to the profile data, it nearly never changes.



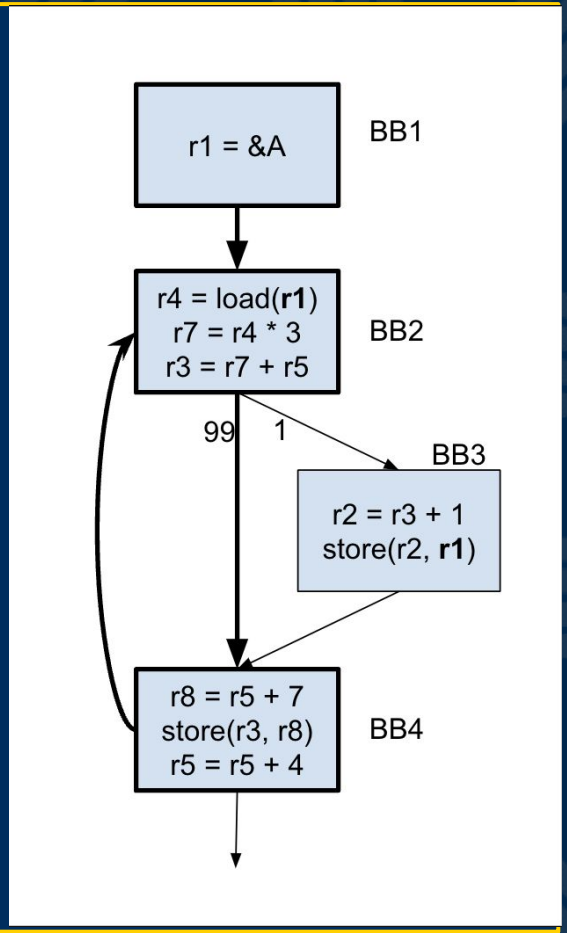
Implementation Steps

- Identify the loops. Then for each loop:
  - Identify the frequent path (≥ 80%).
  - Identify loads that are invariant on the frequent path.
  - Perform LICM on those loads.
  - [Bonus] Perform LICM on other instructions.
  - Add fix-up code to ensure that the execution is correct.

Identify the Frequent Path

- Start at the loop header, keep choosing the branch that is taken at least 80% of the time, or until the loop is closed.
- The cumulative probabilities may drop lower.
- Everything not on the frequent path is considered to be on the infrequent path.

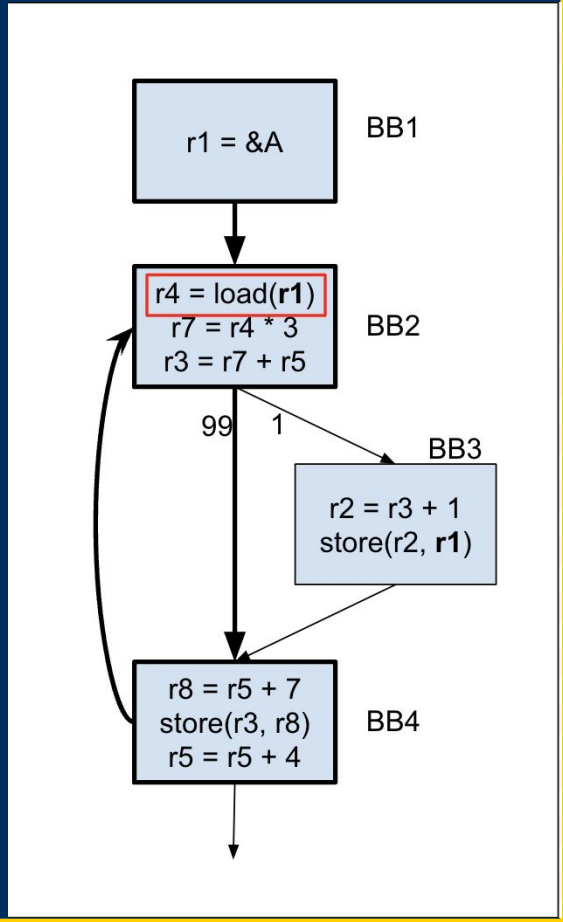
Performance: You can use a different threshold.



Identify the “almost” invariant loads

Now that we consider only the frequent path, the load has become invariant.

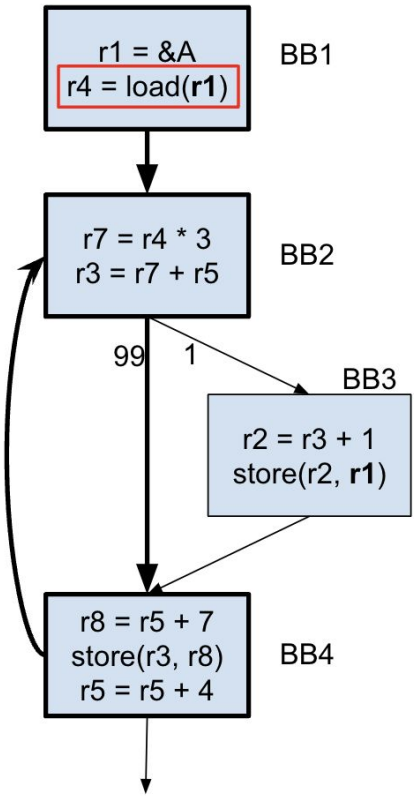
For correctness, you only need to consider the loads.



Move the Load

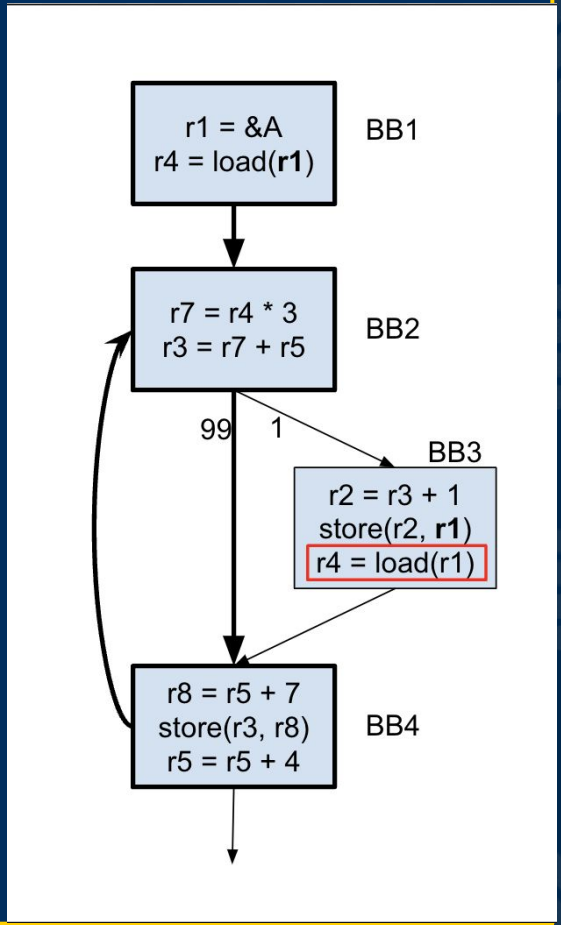
- We can now move the load up to the pre-header.
- This is the key optimization step as the load is now executed only once.

- However it is important to note that the program in its current state will **not be correct**.



Fixing Up

- Now that we have moved the load, we need to add code so that the program execution is correct.
- Just copying the load instruction to the infrequent path will ensure that it works correctly.

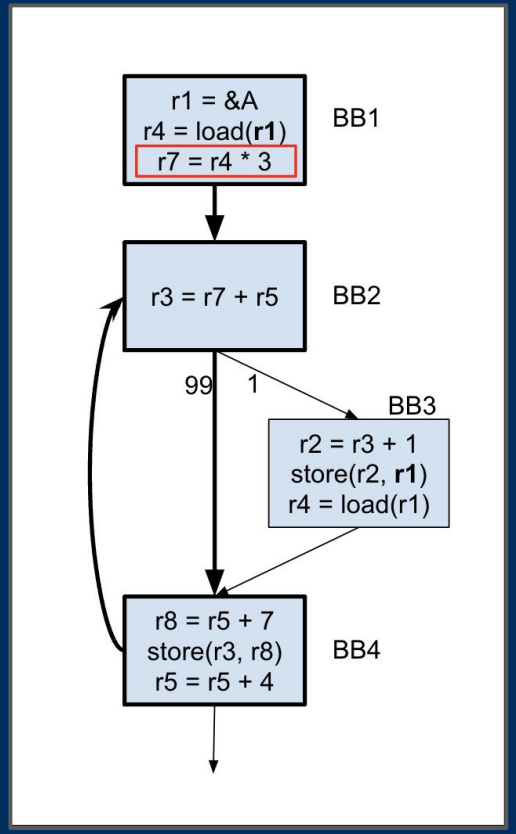


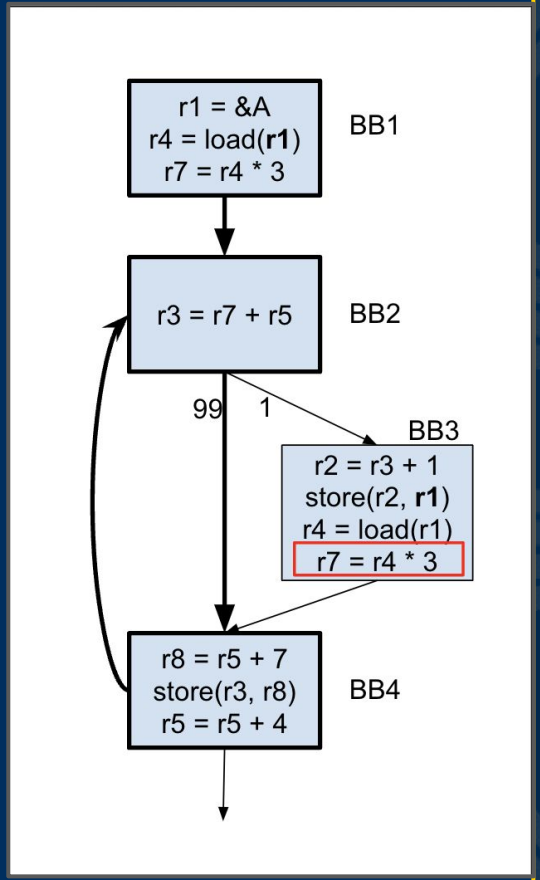
[BONUS] Move more instructions

Since we moved the load, the following instruction has also become invariant in the loop.

So we can move it up to the pre-header as well, adding to our gains.

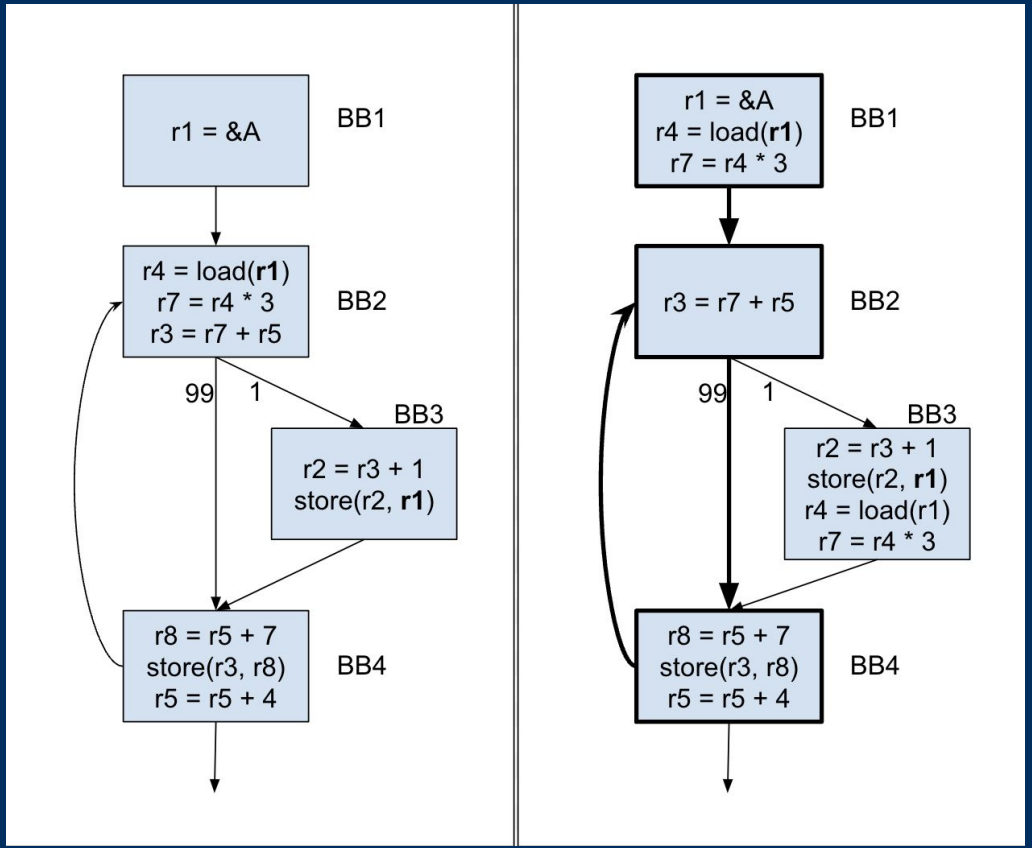
And we need to add the fix up code too.





FPLICM

Before



After

What you have been given

1. A run script
2. A viz script
3. Benchmarks
    a. 6 correctness (Mandatory)
        i. Simple cases exploring different scenarios that your code should be able to handle.
    b. 4 performance (Optional)
        i. Cases with high trip counts and more opportunities for hoisting.
4. Basic template to write code in.

Some LLVM Resources

- Disclaimer: These are only recommendations, you do not have to use these.

Always a useful resource: https://llvm.org/docs/ProgrammersManual.html

Manipulating Basic Blocks

SplitBlock()

- Splits the BB at the specified instruction.

SplitEdge()

- Insert a BB on the edge connecting two specified BBs

Instructions and Variables

- Most Instructions have a constructor (look at the documentation)
  - It allows you to specify operands
  - It also allows you to specify where you want to insert this instruction.
- Many instructions also have a clone() function
- Functions that are useful across all Instructions are in llvm/IR/Instructions.h
  - These include functions that can be used to insert/move instructions
- Use AllocaInst to allocate memory space on the stack.

An important note on SSA

LLVM is in SSA form. You will learn about this on Monday.

This means that when you clone an instruction, the **LHS** will be different.

You need to ensure that the correct values are used in the correct places.

- One solution is to store the value onto the stack (in the pre-header) and retrieve it before any use.
- A better solution is to use PHI nodes to merge the values of the copy and the moved instruction.

Final Notes

- Read the spec and the **Piazza post** carefully and thoroughly.
- Start early.
- Make sure you do not break the program.
- Start with the given script and template.
- If you finish early, attempt the bonus part.
- Check Piazza frequently, someone may have encountered the same issues as you.
- For performance, your code needs to be correct, not just fast.

Thank You