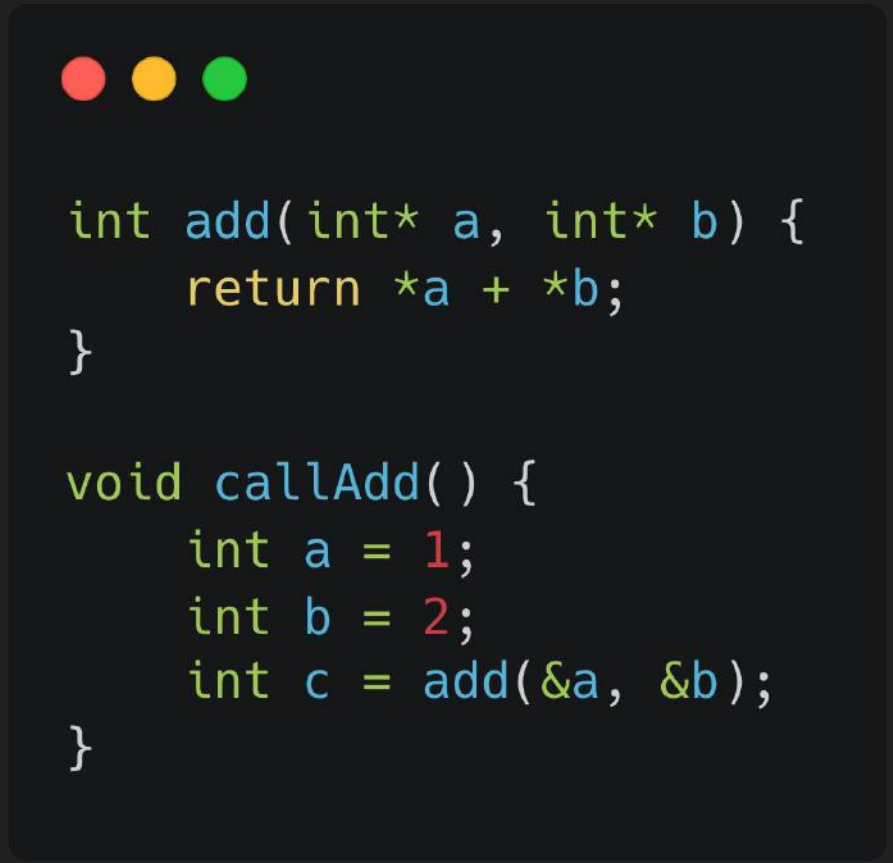
Optimization Research

Sabbir Ahmed

argpromotion

- Implements the argument promotion optimization
- A transform pass that converts function arguments passed “by pointer” to “by value”
- Reduces unnecessary store instructions



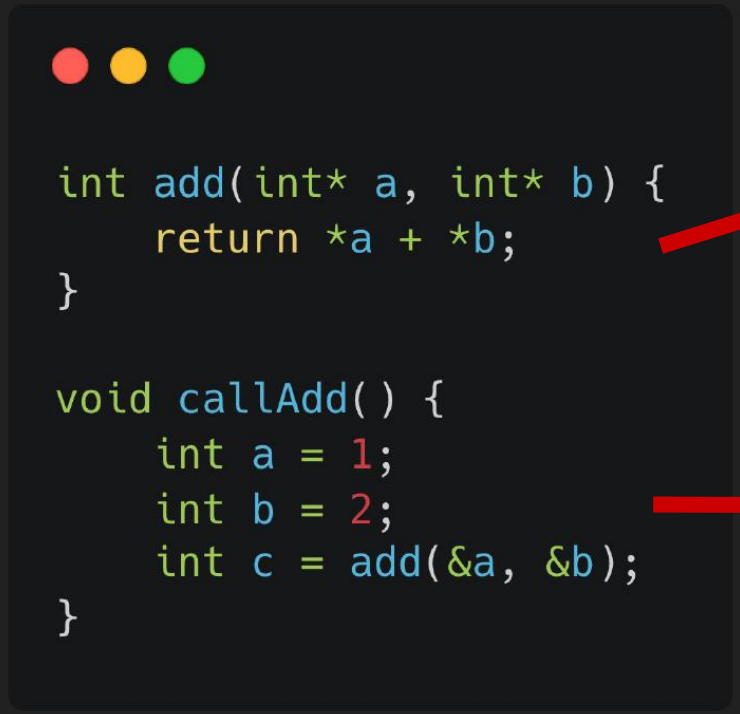
```
int add(int* a, int* b) {
    return *a + *b;
}

void callAdd() {
    int a = 1;
    int b = 2;
    int c = add(&a, &b);
}
```

argpromotion

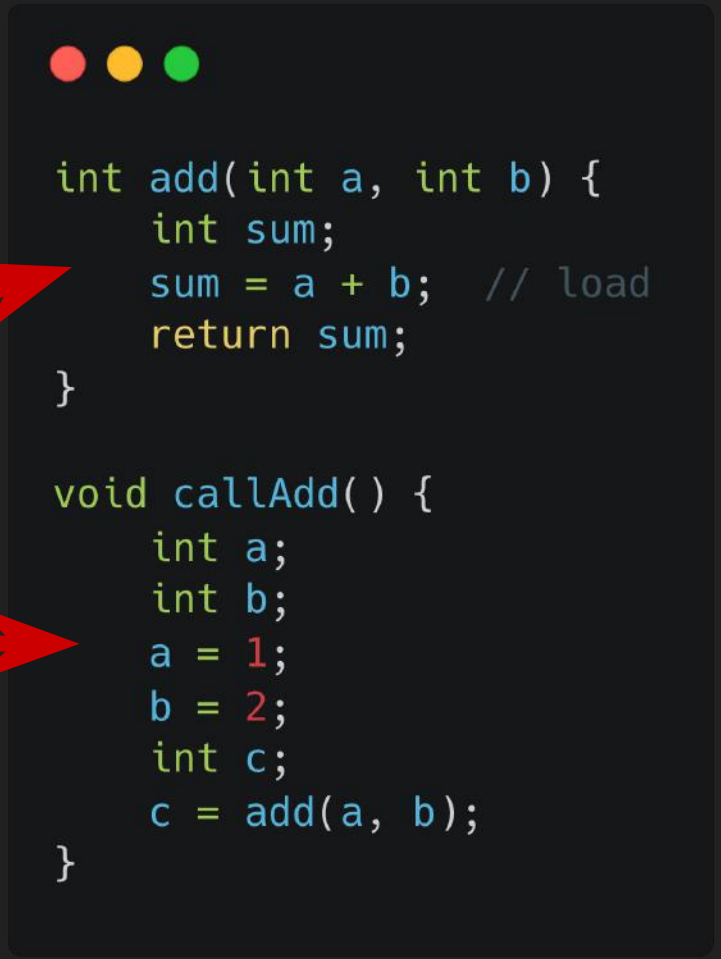
```c
int add(int* a, int* b) {
    return *a + *b;
}

void callAdd() {
    int a = 1;
    int b = 2;
    int c = add(&a, &b);
}
```

```c
int add(int* a, int* b) {
    int tmpa;
    int tmpb;
    tmpa = *a;  // store
    tmpb = *b;  // store
    int sum;
    sum = tmpa + tmpb;  // load
    return sum;
}

void callAdd() {
    int a;
    int b;
    a = 1;
    b = 2;
    int c;
    c = add(&a, &b);
}
```

argpromotion



```
int add(int* a, int* b) {
    return *a + *b;
}

void callAdd() {
    int a = 1;
    int b = 2;
    int c = add(&a, &b);
}
```

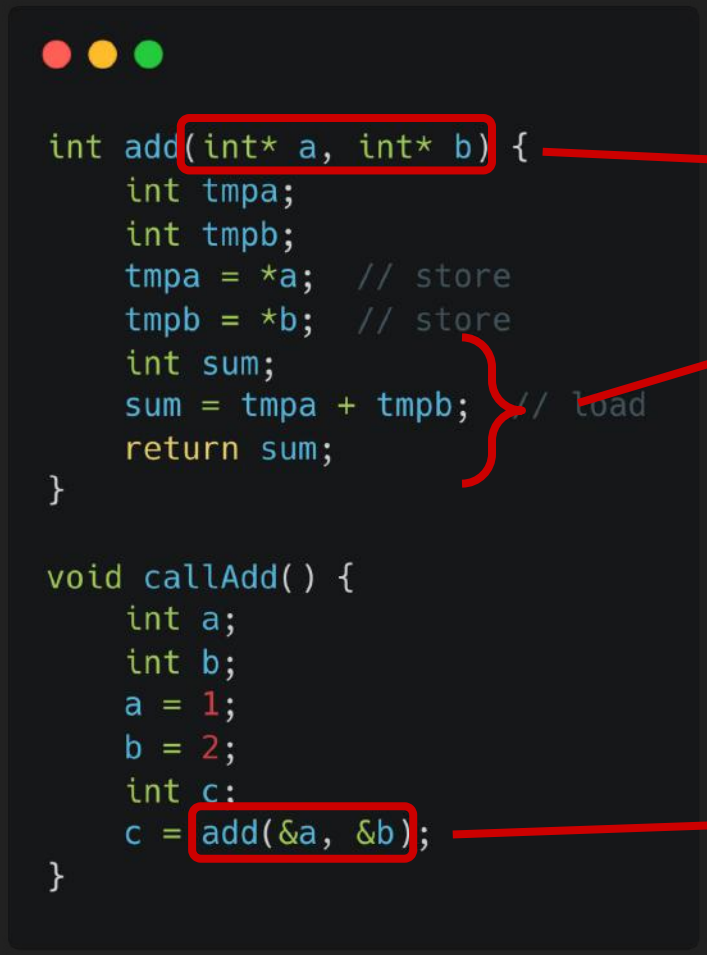


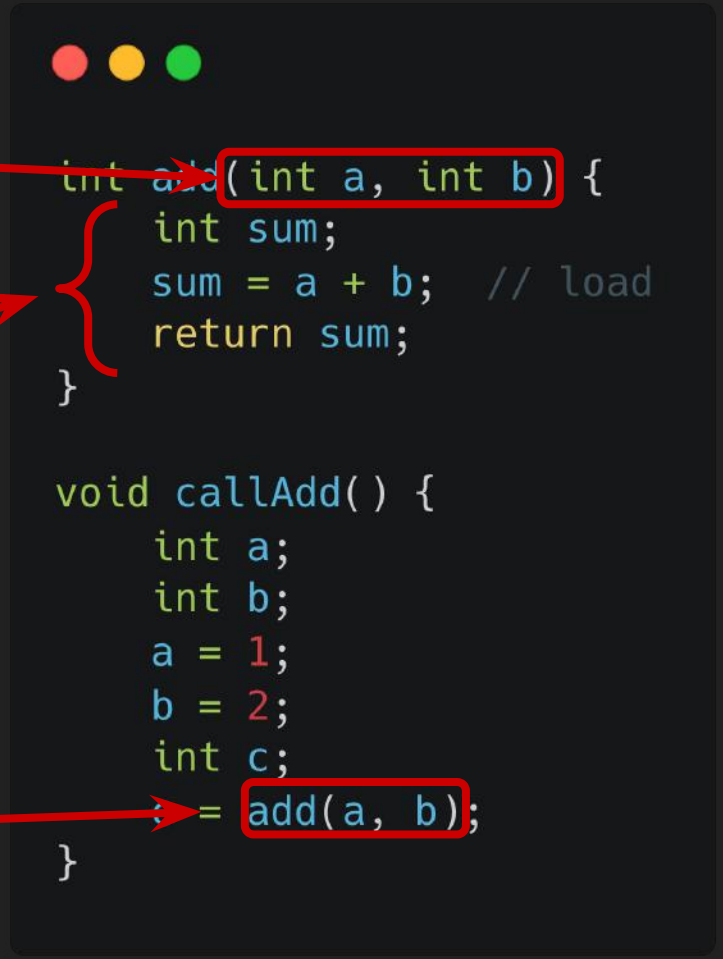
```
int add(int a, int b) {
    int sum;
    sum = a + b;  // load
    return sum;
}

void callAdd() {
    int a;
    int b;
    a = 1;
    b = 2;
    int c;
    c = add(a, b);
}
```

argpromotion

argpromotion

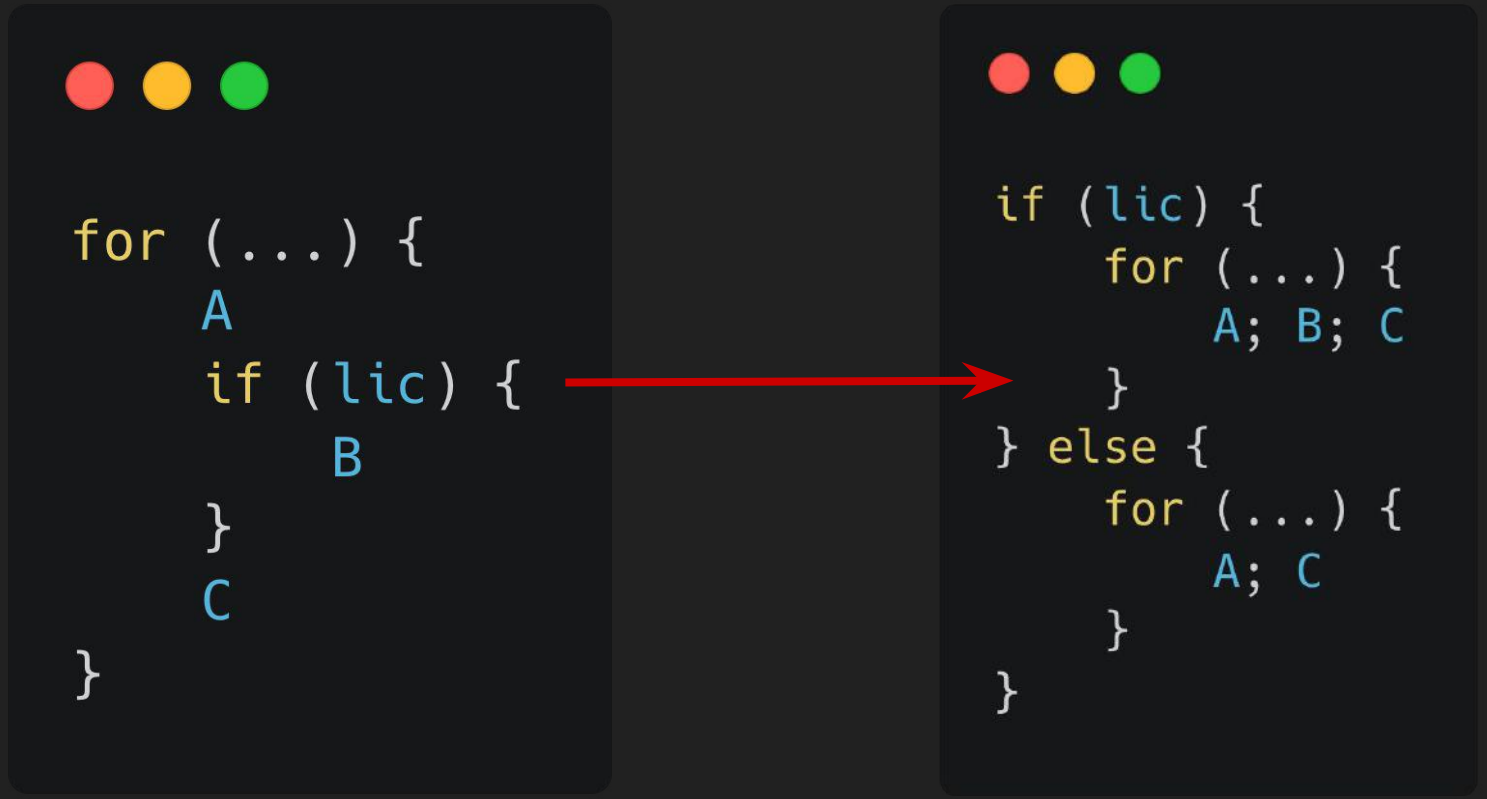
- Change the prototype of the `add` method
- Update all the call sites according to the modified prototype
- Know the reference does not alias other pointers in the `add` method
- Know the loaded values remain unchanged from the function entry to the load
- Know the reference is not being used to store


int add(int* a, int* b) {
int tmpa;
int tmpb;
tmpa = *a; // store
tmpb = *b; // store
int sum;
sum = tmpa + tmpb; // load
return sum;
}
void callAdd() {
int a;
int b;
a = 1;
b = 2;
int c;
c = add(&a, &b);
}


int add(int a, int b) {
int sum;
sum = a + b; // load
return sum;
}
void callAdd() {
int a;
int b;
a = 1;
b = 2;
int c;
= add(a, b);
}

simple-loop-unswitch

- Implements the loop unswitch optimization
- Unswitching: when the loop invariant conditionals inside of the loop gets extracted outside
- Allows for individual loops to be safely parallelized and further optimized



simple-loop-unswitch

simple-loop-unswitch

- The loops duplicate for every conditional branch
- LLVM provides options to determine the form of transformation for this optimization
  - Trivial: when the condition can be unswitched without cloning any code from inside the loop
  - Full: when the branch or switch is completely moved from inside the loop to outside the loop

Thank you!