Compiler Lab Report:
HW8

Name: 韩周吾

ID: 22307130440

Date: 2025.05.21

一、设计逻辑

阶段	作用
simplify	删除度数 < k 的节点，压栈备选
coalesce	合并 move 指令相关节点，减少冗余
freeze	冻结无法合并的 move 节点，便于简化
spill	溢出度数太高的变量，将其分配到内存
select	从栈中弹出并为节点分配实际颜色

二、详细实现分析

1. Simplify 阶段：简化图结构

思路

删除当前图中所有非机器寄存器、且度数 < k 的节点。它们被认为在未来的 select 阶段**一定可以安全着色**，因此可先"压栈"暂存。

```
for (auto &p : graph) {
    if (isMachineReg(node) || isMove(node)) continue;
    if (p.second.size() < k) {
        toSimplify.push_back(node);
    }
}
```

效果

减少图中复杂度，使后续着色更容易，并为栈式回溯建立基础。

---

2. Coalesce 阶段：合并 move 相关变量

思路

合并 `movePairs` 中的相关变量（如 `a = b`），如果合并后不会破坏着色性（即邻居度数仍可接受），就可以安全合并。

```
bool isSafe = true;
for (int neighbor : allNeighbors) {
    if (graph[neighbor].size() >= k) {
        isSafe = false;
        break;
    }
}
```

合并操作包括：

- 修改邻接表，将 `removeNode` 的邻接边转移到 `keepNode`
- 更新 `coalescedMoves` 映射，记录合并来源
- 从 `movePairs` 中移除合并成功的项

效果

减少不必要的 `move` 指令，提高生成代码效率。

---

3. Freeze 阶段：冻结未能合并的 Move 节点

思路

冻结所有参与 move 但当前无法合并的、度数 < k 的节点，将它们视作普通节点用于 simplify。

```
if (isMove(node) && it->second.size() < k) {
    freezeCandidates.push_back(node);
}
```

效果

将瓶颈 move 节点“解冻”，使其参与简化，防止死锁或无法前进的状态。

---

4. Spill 阶段：选择溢出节点

思路

若 simplify、coalesce、freeze 均无可操作节点，选择一个“最差”节点（度数最大）进行溢出处理。此为“软溢出”，真正溢出代码生成在 `select` 阶段后实现。

```
if (degree > maxDegree) {
    maxDegree = degree;
    spillNode = node;
}
```

效果

避免算法卡死在无简化、无可合并、无冻结的死局状态。

---

5. Select 阶段：分配颜色

思路

从 `simplifiedNodes` 栈中弹出节点，为每个节点尝试分配不冲突的颜色（寄存器编号）。如果找不到合法颜色，则标记为 `spilled`。

同时对合并节点 `coalescedMoves` 分配与其主节点相同的颜色。

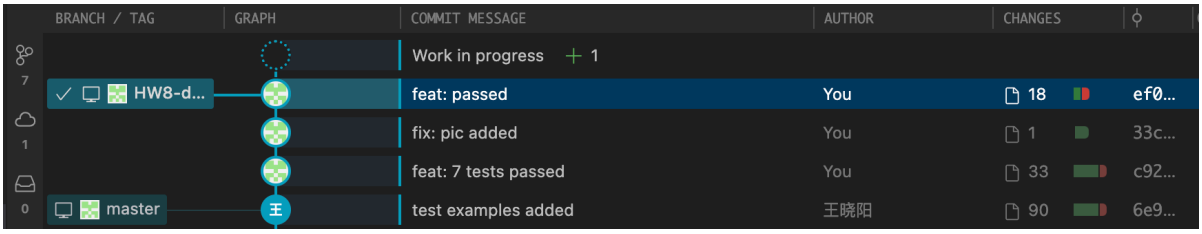
```
for (int color = 0; color < k; color++) {
    if (usedColors.find(color) == usedColors.end()) {
        chosenColor = color;
        break;
    }
}
```

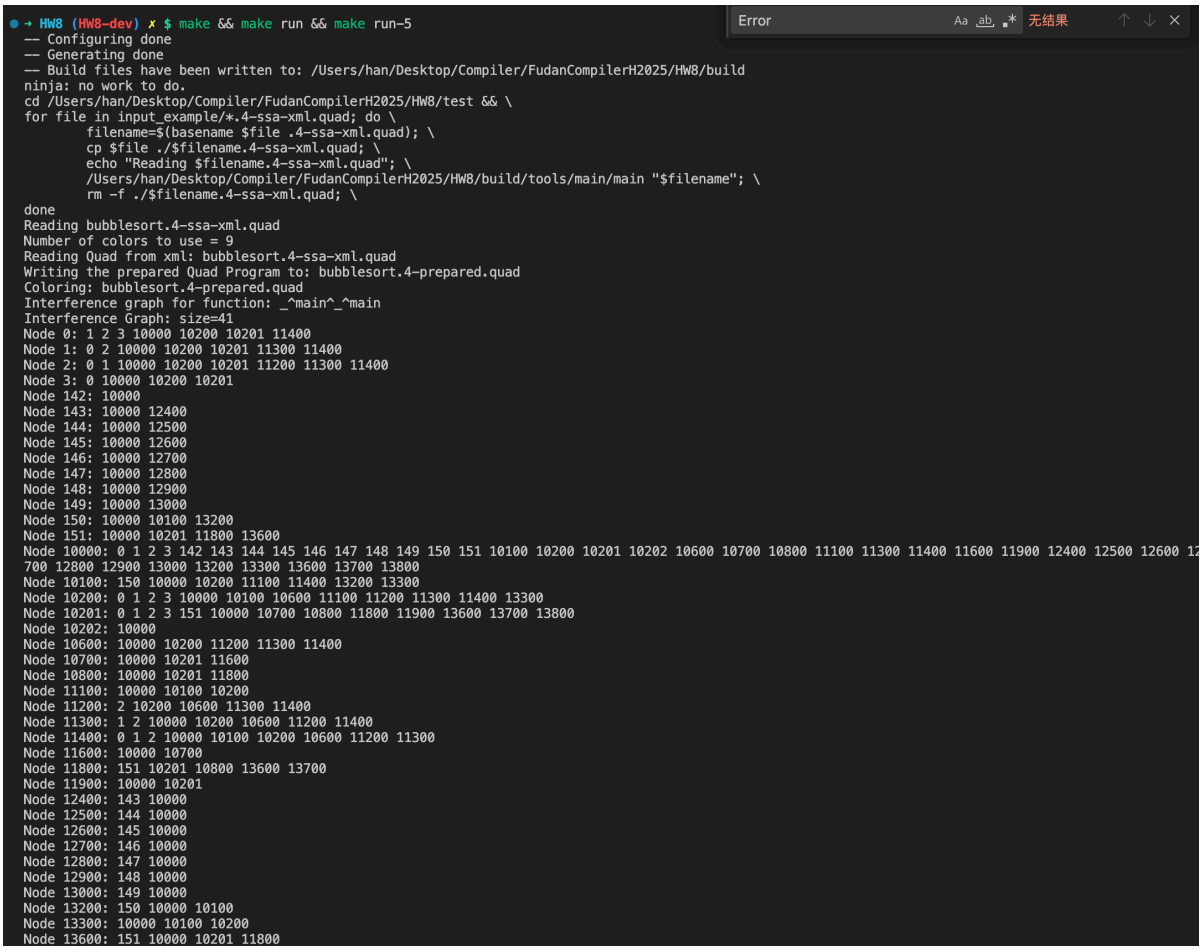
最后使用 `checkColoring()` 验证所有冲突节点着色是否合法。

效果

实现最终着色方案，输出分配结果。

Graphs and Figures




Error
无结果
➜ HW8 (HW8-dev) ✗ $ make && make run && make run-5
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/han/Desktop/Compiler/FudanCompilerH2025/HW8/build
ninja: no work to do.
cd /Users/han/Desktop/Compiler/FudanCompilerH2025/HW8/test && \
for file in input_example/*.4-ssa-xml.quad; do \
filename=$(basename $file .4-ssa-xml.quad); \
cp $file ./$filename.4-ssa-xml.quad; \
echo "Reading $filename.4-ssa-xml.quad"; \
/Users/han/Desktop/Compiler/FudanCompilerH2025/HW8/build/tools/main/main "$filename"; \
rm -f ./$filename.4-ssa-xml.quad; \
done
Reading bubblesort.4-ssa-xml.quad
Number of colors to use = 9
Reading Quad from xml: bubblesort.4-ssa-xml.quad
Writing the prepared Quad Program to: bubblesort.4-prepared.quad
Coloring: bubblesort.4-prepared.quad
Interference graph for function: _^main^_^main
Interference Graph: size=41
Node 0: 1 2 3 10000 10200 10201 11400
Node 1: 0 2 10000 10200 10201 11300 11400
Node 2: 0 1 10000 10200 10201 11200 11300 11400
Node 3: 0 10000 10200 10201
Node 142: 10000
Node 143: 10000 12400
Node 144: 10000 12500
Node 145: 10000 12600
Node 146: 10000 12700
Node 147: 10000 12800
Node 148: 10000 12900
Node 149: 10000 13000
Node 150: 10000 10100 13200
Node 151: 10000 10201 11800 13600
Node 10000: 0 1 2 3 142 143 144 145 146 147 148 149 150 151 10100 10200 10201 10202 10600 10700 10800 11100 11300 11400 11600 11900 12400 12500 12600 12
700 12800 12900 13000 13200 13300 13600 13700 13800
Node 10100: 150 10000 10200 11100 11400 13200 13300
Node 10200: 0 1 2 3 10000 10100 10600 11100 11200 11300 11400 13300
Node 10201: 0 1 2 3 151 10000 10700 10800 11800 11900 13600 13700 13800
Node 10202: 10000
Node 10600: 10000 10200 11200 11300 11400
Node 10700: 10000 10201 11600
Node 10800: 10000 10201 11800
Node 11100: 10000 10100 10200
Node 11200: 2 10200 10600 11300 11400
Node 11300: 1 2 10000 10200 10600 11200 11400
Node 11400: 0 1 2 10000 10100 10200 10600 11200 11300
Node 11600: 10000 10700
Node 11800: 151 10201 10800 13600 13700
Node 11900: 10000 10201
Node 12400: 143 10000
Node 12500: 144 10000
Node 12600: 145 10000
Node 12700: 146 10000
Node 12800: 147 10000
Node 12900: 148 10000
Node 13000: 149 10000
Node 13200: 150 10000 10100
Node 13300: 10000 10100 10200
Node 13600: 151 10000 10201 11800