SAARLAND UNIVERSITY
COMPUTER SCIENCE

Compiler Construction (WS 2017/18)

Exercise Sheet 10

Compiler Design Lab
Prof. Dr. Sebastian Hack
Fabian Ritter, B.Sc.

Exercise 10.1 Colorability

Show that Chaitin's algorithm finds a k-coloring for every k-colorable chordal graph.
Hint: Every chordal graph with at least 2 *nodes has* 2 *simplicial nodes.*

Exercise 10.2 Partitioned Boolean Quadratic Programming

Prove that finding a solution for a PBQP problem to be NP-hard by reducing SAT to PBQP.
Hint: Reconsider the NP-hardness proof for instruction selection. First, try to map the boolean formula $(a \wedge b) \vee \neg b$ from the example in Figure 2 of Koes' paper[1] to PBQP. Then, you can derive an algorithm to map any SAT problem to PBQP. Generally, to map $a \vee b$ you will need four nodes: one for a, one for b, one for $\vee$ and an auxiliary node.

Exercise 10.3 PBQP Applied

1. Study the LLVM-IR program below and draw the value graph for the loop body (`for.body`). Include constants, function arguments and `PHI` nodes from other blocks in the graph. Futhermore, replace the `getelementptr` instruction by appropriate scalar operations (`add`/`mul`) and fold constant expressions together. Assume the size of an `i32` is 4 bytes.

2. Use the patterns on the PBQP slide 19 and the costs shown below to create a PBQP instance for the graph constructed in part 1. Assume the patterns *AC* and *A* are also available for multiplications (*MC*/*M*) and that there is a pattern *AR* for accessing an arguments.

Pattern	AR	C	P	A	AC	M	MC	L	LA	LAC
Cost	5	10	15	25	30	35	40	100	100	100

3. Use the optimality-preserving reductions and the heuristic reduction to find a solution for the PBQP problem. Write down the order in which edges/nodes are eliminated and the rule that was applied.

```
define i32 @array_sum(i32* %A, i32* B, i32 %N) {
entry:
  br label %for.cond

for.cond:
  %iv = phi i32 [ 0, %entry ], [ %iv.inc, %for.body ]
  %sum = phi i32 [ 0, %entry ], [ %add1, %for.body ]
  %B.cur = phi i32* [ %B, %entry ], [ %B.idx, %for.body ]
  %cmp = icmp slt i32 %iv, %N
  br i1 %cmp, label %for.body, label %for.end

for.body:
  %A.idx = getelementptr i32, i32* A, i32 %iv
  %B.idx = getelementptr i32, i32* B.cur, i32 1
  %A.val = load i32, i32* A.idx, align 4
  %B.val = load i32, i32* B.idx, align 4
  %add1 = add i32 %sum, %A.val
  %add2 = add i32 %add1, %B.val
  %iv.inc = add i32 1, %iv
  br label %for.cond

for.end:
  ret i32 %sum
}
```

[1] available at `https://dl.acm.org/citation.cfm?id=1356065` (from university network)

Project task G Implement Sparse Conditional Constant Propagation

For this project assignment, we introduce an additional compiler switch, `--optimize`. This switch performs the same operation as `--compile` and additionally performs optimizations. Implement the following optimization:

- Perform the SCCP [2] analysis.
- If applicable,
  - substitute instructions with constants,
  - substitute conditional branches with unconditional ones,
  - remove dead instructions and
  - remove unreachable blocks.
- If you do not want to implement SSA construction yourself, you may use the LLVM *mem2reg* pass (see `llvm::createPromoteMemoryToRegisterPass()`) **but no other LLVM passes**.
- This is the last project assignment. Information on scheduling your code review meeting with us will be available in the forum.
- The soft deadline for this milestone is 2018-02-02.
- Keep it simple!

[2] consider `https://dl.acm.org/citation.cfm?id=103136` for more details if you haven't already (accessible from the university network).