Unit Project Report
CS 425 Compiler Construction

Lingyi Liu (liu187), Kuan-Yu Tseng (ktseng2)

December 15, 2012

1 Chaitin-Briggs Algorithm

Chaitin-Briggs algorithm[1] is a register allocation algorithm that utilizes graph coloring on the interference graph that are derived from the live ranges of registers, to allocate physical register for each virtual register. There are 4 major steps for this algorithm:**Live Range Computation**, **Interference Graph Construction**, **Spill Cost Calculation**, **Graph Coloring**. These steps are described in detail in the following.

1.1 Live Range Computation

The Global Live Ranges of a virtual register *vreg* is a partition of the references (definitions or uses) of *vreg*. If one definition *def* of *vreg* is in the Live Range *lr*, then all uses reachable from *def* are also in *lr*. If one use of *vreg* is in the Live Range *lr*, then all defs that reaches the use are also in *lr*. Using the Live Variable Analysis, we can compute the def-use chain for each virtual register. And therefore we can compute the live ranges with `union-find` algorithm.

1.2 Interference Graph Construction

A register *vreg1* is said to interfere with another register *vreg2* if *vreg1* is defined when *vreg2* is live. Therefore, to show the interference among all registers, we can construct a interfernce graph where each node represents a register and each undirected edge represents whether the nodes on both ends interfere with each other. The algorithm for constructing the interfence graph is as follows. We need the result of Live Variable Analysis to be able to get LiveOut for each basic block.

Algorithm 1 Interference Graph Construction Algorithm

$build\_interference\_graph(Func)$
1: Intialize IG to a graph with each register as a node and no edge
2: **for each** $BB \in Func$ **do**
3: $LiveNow = BB.getLiveOut()$
4: **for each** $Instr : (a \text{ op } b \rightarrow c) \in BB$ (in the order from bottom to top) **do**
5: IG.addEdge(c, a)
6: IG.addEdge(c, b)
7: $LiveNow.remove(c)$
8: $LiveNow.add(a)$
9: $LiveNow.add(b)$
10: **end for**
11: **end for**
12: **return** IG

1.3 Spill Cost Estimation

The most important part in the Chaintin-Briggs algorithm is to estimate spill cost because a precise estimation of the cost will enable the algorithm to pick the right register to spill, which may greatly improve the performance. There are two ways to spill a register: one way is *load/store*, which uses a stack slot to store

the register after each definition and load the register before each use. Another way is *rematerialization*, which recomputes the value in the register. Each way may incurs less spill cost than the other depending on the situations. Another factor of spill cost is the execution frequencies. We tend to spill registers that have least execution frequencies. We can weight a register by 10^d where d is the loop depth or by branching probability. Or we can profile the program with some representative inputs to get better estimation of the execution frequencies.

1.4 Graph Coloring

2 Heuristic Approach for Spilling

3 Project Status

3.1 What is Working?

3.2 What is Not Working?

3.3 Potential Improvement

4 Experimetnal Results

4.1 Benchmark Programs

4.2 Execution Time

4.3 Number of Sills

References

[1] G. J. Chaitin, "Register allocation & spilling via graph coloring," in *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, ser. SIGPLAN '82. New York, NY, USA: ACM, 1982, pp. 98–105. [Online]. Available: http://doi.acm.org/10.1145/800230.806984