Code Instruction Selection Based on SSA-Graphs

Sumant Sathe
sumant [AT] isy.liu.se

1. Introduction
This is a summary of the paper “Code Instruction Selection Based on SSA-Graphs” by Eckstein, et. al.
The instruction selector translates the intermediate representation into target code.
Tree pattern matching is a widely used technique for instruction selection. Usually the unit of
translation is a statement which is represented as a DFT. A set of rules is used to match the DFT.
The matcher selects those rules such that the sum of all applied rule costs is minimum. An algo-
rithm for tree pattern matching has two phases : labeling and reducing. In the labeling phase mini-
mal costs are calculated for each node and non-terminal. In the reduction phase the tree is
traversed top-down and the rules with minimal costs are selected.
DAG matching is an extension to tree matching. Instead of trees, DAG’s are considered. DAG
matching is an NP-complete problem. This algorithms does code duplication. This algorithm does
not work on a graph containing cycles.
Both tree pattern matching and DAG matching do not consider the computational flow of func-
tions. This paper presents a new technique for instruction selection of code generators. The com-
putational flow of a whole function is taken into account. For representing the computational flow
the SSA-graph is used which combines DFT’s and def-use relations of a function.

2. The Algorithm
The starting point of the technique is the SSA-graph. An ambiguous grammar describes possible
derivations of the SSA-graphs. Production rules have cost terms and code templates. Cost terms
are used to find the derivation with minimum overall cost. Parsing SSA-graphs is NP-complete.
To overcome this problem, the instruction selection problem for SSA-graphs is mapped to the par-
titioned boolean quadritic problem (PBQP).
First, the SSA-graph with its ambiguous grammar is mapped to PBQP. Second, the PBQP solver
computes the grammar derivation with minimum cost. Third, based on the grammar derivation,
code is produced.
The PBQP solver consists of two phases : In the first phase the graph is reduced until a trivial
solution remains. In the second phase the solution is back-propagated.

3. Matching Problem
The grammars used for matching SSA-graphs are similar to grammars used by tree pattern mat-
tchers. The mapping of the SSA-graph matching problem to PBQP is now described. In the first
step of the mapping, the grammar is transformed to normal form. A grammar is in normal form if
there are only production rules which are either base rules or chain rules. For solving the matching problem, the PBQP solver is employed. The PBQP problem is defined as follows:

\[
\min f = \left[ \sum_{1 \leq i < j \leq n} x_i \cdot C_{ij} \cdot x_j^T \right] + \left[ \sum_{1 \leq i \leq n} c_i \cdot x_i^T \right]
\]

subject to: \( \forall i \in 1 \ldots n : x_i \cdot 1^T = 1 \)

where \( x_i \) are boolean vectors for which only one element is set to one and \( n \) is the number of vectors. \( C_{ij} \) are cost matrices, and \( c_i \) is a cost vector. For each boolean vector \( x_i \), an element has to be chosen such that the objective function \( f \) becomes a minimum.

The mapping from the SSA-graph matching problem is done in three steps: (1) construct the PBQP graph based on the SSA graph, (2) determine cost-vectors of the nodes, and (3) determine cost-matrices of edges.

4. PBQP Solver
In the first phase reduction rules are applied to nodes with degree one and two (ReduceI and ReduceII reductions). If the reduction with ReduceI and ReduceII is not possible, a heuristic must be applied (ReduceN reduction). The heuristic selects the local minimum for the chosen node and eliminates the node. The reduction process is performed until a trivial solution remains, i.e. nodes with degree zero are left. In the second phase, the graph is reconstructed in reverse order of the reduction phase and the solution is back-propagated.

In addition, simplification reductions are performed. They are (1) elimination of nodes which have only one cost vector element and (2) elimination of independent edges.

After reduction only nodes with degree zero remain and the rules can be selected by finding the index of the minimum vector element. The rules of all other nodes can be selected by reconstructing the PBQP graph in reverse order of reductions. In each reconstruction step one node is reinserted into the graph and the rule of this node is selected. Selecting the rule is done by choosing the rule with minimum cost for the node. This can be done, because the rules of all adjacent nodes are already known.

5. Experimental Results
A number of DSP benchmarks like AAC, MPEG, GSM, etc. have been used for experiments. The quality of the solution depends on the density of the PBQP graphs. If a graph can be reduced with ReduceI and ReduceII rules, the solution is optimal. An important observation is that only a small fraction of all nodes are ReduceN nodes (less than 1%). Therefore the solution obtained from the PBQP solver is near optimal. An effective way to improve the solution is to recursively enumerate the first ReduceN nodes in a graph. In many graphs only a few ReduceN nodes exist and by moderate enumeration an optimal solution can be achieved.

The performance gain of a SSA-graph matcher compared to a tree pattern matcher is significant (up to 82%) in comparison to classical tree pattern matching methods. The results were obtained without modifying the grammar. Though the overhead of the PBQP solver is higher than tree pattern matching methods, the compile time overhead is in acceptable bounds (for a huge number of enumerations, it is significantly higher).