Conference contribution:
The optimisation problem

Within the field of compilers, the term optimisation is something of a misnomer. During the compilation process, compilers attempt to improve (with respect to both size and performance) the sequences of machine-level instructions it generates by applying a fixed set of transforms, reductions and equivalences. In many modern compilers, this can result in significant improvements, but it is unlikely to produce optimal sequences of instructions; and if it does, it will not be possible to determine that they are indeed optimal.

In a significant range of applications, this approach to code generation is not sufficient; examples include resource-critical environments such as embedded domains, optimising compilers for the increasingly complex modern computer architectures and high-performance computing.

Superoptimisation is an approach in which code generation for acyclic code sequences as a combinatorial search problem. Rather than starting with crudely generated code and improving it, a superoptimizer starts with the specification of a function and performs a directed search for a sequence of instructions that meets this specification. Superoptimisation provides a fresh approach to the optimisation problem by aiming for optimality from the outset.

Solution: TOAST

The Total Optimisation using Answer Set Technology (TOAST) [2,3] system uses ASP as the modelling and computation framework to solve the superoptimisation search problem. The motivation for the TOAST system is as follows:

- New structured approach to optimisation
- Lack of proven optimality of existing techniques
- Emergence of new performance-critical domains
- Modelling and computational power of ASP

TOAST consists of modular interacting components that generate AnsProlog programs, starting with a model of the microprocessor architecture, its instruction semantics and the original sequence to be optimised. A controlling user interface utilises these components to generate a shorter, superoptimised version of the original sequence using off-the-shelf domain solving tools.

Experimental Results

The sequence5 test is a sequence that is already optimal, giving an approximate ceiling on the performance of any system in searching over the large instruction space. verifytest1 tests the (non-trivial) equivalence of two code sequences, while verifytest2 tests the (non-trivial) equivalence of two code sequences that only differ on one set of inputs. The table below presents timings for these search and verification tests for the SPARC V8, a popular 32-bit RISC architecture; solver time outs occurred after 200 hours.

<table>
<thead>
<tr>
<th>Solver</th>
<th>sequence5</th>
<th>verifytest1</th>
<th>verifytest2</th>
</tr>
</thead>
<tbody>
<tr>
<td>clasp-1.2.1</td>
<td>0.28</td>
<td>2.01</td>
<td>1187</td>
</tr>
<tr>
<td>smodels-3.97</td>
<td>0.37</td>
<td>8.69</td>
<td>1019</td>
</tr>
<tr>
<td>models-3.33</td>
<td>0.28</td>
<td>7.57</td>
<td>1622</td>
</tr>
<tr>
<td>smodels-ie-1.0.0</td>
<td>0.21</td>
<td>6.91</td>
<td>4870</td>
</tr>
<tr>
<td>sup-0.4</td>
<td>0.44</td>
<td>3.15</td>
<td>12583</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solver</th>
<th>sequence5</th>
<th>verifytest1</th>
<th>verifytest2</th>
</tr>
</thead>
<tbody>
<tr>
<td>clasp-1.2.1</td>
<td>0.28</td>
<td>2.01</td>
<td>1187</td>
</tr>
<tr>
<td>smodels-3.97</td>
<td>0.37</td>
<td>8.69</td>
<td>1019</td>
</tr>
<tr>
<td>models-3.33</td>
<td>0.28</td>
<td>7.57</td>
<td>1622</td>
</tr>
<tr>
<td>smodels-ie-1.0.0</td>
<td>0.21</td>
<td>6.91</td>
<td>4870</td>
</tr>
<tr>
<td>sup-0.4</td>
<td>0.44</td>
<td>3.15</td>
<td>12583</td>
</tr>
</tbody>
</table>

Conclusions and Future Work

- Development of a structured approach and adaptable framework to generating truly optimal code sequences is an important development for the domain.
- Superoptimisation of code is achievable in the general case and can be used to generate provably optimal code sequences for 32-bit architectures (and that do the same for 64-bit architectures as well).
- ASP is an appropriate paradigm for reasoning about large-scale, real-world problems. The flexibility of AnsProlog allows arbitrary constraints to be added to the search with minimal effort, something that is very difficult in the case of procedural superoptimisers.
- With further advances in solver technology and search heuristics, it is hoped that TOAST can be built into a competitive superoptimising system, especially for use as a peephole superoptimizer, via the generation of equivalence classes of code sequences with base/dupe/-inline.
- Key future applications areas would be in compiler toolchains such as GCC and JIT compilers, along with extensions to the modelling framework to handle multi-threaded and multi-core architectures. Also, focusing on the embedded domain, such as the ARM family of processors.

Answer Set Programming

A Answer Set Programming (ASP) [1] is a declarative programming paradigm that allows reasoning about possible word views in the absence of complete information. It is a powerful and intuitive non-monotonic logic programming language for modelling, reasoning and verification tasks.

ASP describes a problem as a logic program in AnsProlog, a set of axioms and a goal statement, under the answer set semantics of logic programming in such a way that solving the problem is reduced to computing the answer sets of the program.

Due to the increasing efficiency of its heuristic domain tools, known as solvers (such as clasp, smodels, smodels.ie and sup), ASP is particularly suited to difficult (primarily NP-hard) search problems, making a number of problems tractable in the general case.

Example applications of ASP to real-world problems include diagnostic reasoning, multi-agent systems, phylogenetics, biological networks, automatic music composition, evolutionary history of languages, cryptography, security engineering, instruction scheduling, program analysis and decision support systems for the NASA Space Shuttle.

TOAST Framework

Superoptimisation naturally decomposes into two sub-problems: searching for sequences that meet specific criteria and then verifying which of these candidates are functionally equivalent to the original sequence.

The TOAST system is currently able to superoptimise sequences of five instructions in a practical time with current solving tools. This is a significant result considering empirical evidence for the average size of basic blocks (between 5-6 instructions). This can also be extended to superoptimise superblocks of instructions.

References