Domain-Specific Optimisation

with

User-Defined Rules

in

CodeBoost

Otto Skroven Bagge
Magne Haveraaen

RULE 2003
What is CodeBoost?

- A framework for source-to-source transformation of C++ programs
  - Supports significant subset of C++, including function and operator overloading, and templates

- Primarily intended to support the Sophus numerical library
  - Domain-specific optimisation

- Written in the Stratego program transformation language — but the Sophus developers shouldn’t need to learn about Stratego and CodeBoost internals
User-friendly specification of domain-specific optimisations

Optimisation rules should be easy to specify for people without intimate knowledge of program transformation, CodeBoost and Stratego.

- Concrete syntax
  - Stratego’s concrete syntax won’t work with the current C++ parser

- Embedded rules
  - should be possible to specify optimisations within the C++ program, together with relevant parts of the library

- Easy matching of calls to overloaded functions
  - shouldn’t need to specify complete function signature in the match pattern
Anatomy of a Rule

```c++
void rules()
{
    int x;
    simplify: pow(x, 2) = y * y, y = tmp(x);
}
```

- Syntactically valid C++ code, interpreted as rules by CodeBoost
- Rules are contained within `rules()` functions
- Local variables are meta-variables
- Conditions follow after comma; can call other rules or builtins
- Rules with predefined names such as `simplify`, `topdown`, `bottomup`, etc. will be applied by the appropriate transformation modules
More features

• List matching — for functions accepting a variable number of arguments:

\[
\text{simplify: } f(_\text{list}_\text{(a)}, g(b), _\text{list}_\text{(c)})
= fg(b, _\text{list}_\text{(a)}, _\text{list}_\text{(c))};
\]

• Generic rules, in which the function name is also a meta-variable:

```cpp
void rules()
{
    T (*f)(T, T); // declare f as function pointer
    T x, y;
    commute: f(x, y) = f(y, x), commutative(!f(x, y));

    int a, b;
    commutative: (a + b) = true;
}
```
How does it work?

- Long pipeline of modules, working on abstract syntax tree
- Semantic analysis annotates all calls with their corresponding function signatures, uniquely identifying the called function
- After analysis, make-rules picks up the rules and stores them alongside the AST
- Rules are applied by transformation modules — the exact sequence of transformation modules is specified by the user
- Rule interpreter is written in Stratego, and makes user-defined rules available as Stratego rules
Application: Index optimisations for Sophus

- Sophus uses a generic map function for operating on huge indexed data structures (meshes). The abstract, generic nature of the map function makes it prohibitively slow.

- User-defined rules are used to:
  - Inline calls to overloaded index operators
  - Remove redundant translations between mesh indexing (multi-dimensional) and C++ array indexing (single integer)

- Results are impressive:

<table>
<thead>
<tr>
<th></th>
<th>Not optimised</th>
<th>Basic</th>
<th>Idx opt.</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>827.0s</td>
<td>629.9s</td>
<td>110.5s</td>
<td>5.7</td>
</tr>
<tr>
<td>Large</td>
<td>25435s</td>
<td>19028s</td>
<td>3996s</td>
<td>4.8</td>
</tr>
</tbody>
</table>
Index Optimisation Example

Mesh M; Point P; Shape S; int i;

inline: M[P] = M.data[getlex(P)];
simplify: getlex(setlex(S,i)) = i;

for(i = 0; i < N; i++)
    A.data[i] = f(B[setlex(S,i)], C[setlex(S,i)]);
->
for(i = 0; i < N; i++)
    A.data[i] = f(B.data[getlex(setlex(S,i))],
               C.data[getlex(setlex(S,i))]);
->
for(i = 0; i < N; i++)
    A.data[i] = f(B.data[i], C.data[i]);
Future plans

• Develop better strategies for domain-specific optimisation

• Combine with dataflow analysis
  – Use analysis results in conditions
  – For variables, do matching either on the variable itself, or on its propagated value
Conclusion

• User-defined rules
  – are written in concrete syntax, within C++ programs
  – allow easy matching on semantic information — semantic analysis fills in correct signature and type annotations
  – support conditions and list matching
  – support several different strategies (but not user-definable strategies)
  – provide a convenient way of specifying domain-specific optimisations
• CodeBoost is Free Software (GPL)

• Source code and more information is available at

http://www.codeboost.org/

• Thanks to: Eelco Visser, Karl-Trygve Kalleberg and May-Lill Sande for help and inspiration, and to Chr. Michelsen Research AS and the Research Council of Norway (NFR) for financial support and computing resources.