Automated generation of templated functions to improve code readability and performance

TECHNISCHE UNIVERSITÄT DARMSTADT

Applicable for students as HiWi, Bachelor of Science, Master of Science
Keywords: *Clang, Source Analysis, Source Transformation, Software Quality*

SCIENTIFIC COMPUTING

Tim Heldmann
tim.heldmann@tu-darmstadt.de

Office: S1|03 Room 4a
Hochschulstraße 1
64283 Darmstadt
Tel. 06151 16-27275

Date: 16th November, 2022

Introduction

Templating is a C++ feature that mainly allows the compiler to generate type specific version of function if given a template description for the function.
This feature can also be used to improve code that makes use of polymorphic objects. Instead of resolving a function call on a polymorphic object at runtime, the object itself is a template specialization of the requested container functions. While this does not give all the benefits of polymorphism, it can be faster.

Task

Develop a Clang-based refactoring tool, that automatically detects function with identical instruction content but differing input parameters, and automatically generate a template description for this function. Replace the uses of all functions found to now use the template function. Research the extent to which this approach can be used to handle runtime resolved calls to functions of polymorphic objects, and implement a proof-of-concept of this functionality in the refactoring tool.

```
1  int square_int(int a) {return a*a;}
2  double square_double(double b) {return b*b;}
```

Figure 1: Two functions with identical instructions but differing function signiture

```
1  template<typename T>
2  T square_T(T value){return value*value;}
```

Figure 2: A template version of the above functions, which will be compiler instantiated

What you will be doing

- **(a)** Develop a Clang refactoring tool, which detects transformable functions, and generates a templated version of this function accordingly
- **(b)** Evaluate this approach w.r.t. applicability and commonness of occurrence
- **(c)** Extend this approach to reduce or remove polymorphic function calls via templating
- **(d)** Evaluate the performance changes of the de-polymorphed code

Qualifications

- Knowledge of the Clang tooling library [1].
- Experience with modern C++, especially the template construct and polymorphism [2].

References

[1] https://clang.llvm.org/docs/LibTooling.html

[2] https://en.wikipedia.org/wiki/Curiously_recurring_template_pattern