Grafter: A Clang tool for tree traversals fusion

Laith Sakka   Kirshanthan Sundararajah   Milind Kulkarni
What is Grafter?

Grafter performs source to source transformations to fuse recursive functions that traverse trees

\[
\text{computeHeight(\text{render\_tree})}; \\
\text{computeWidth(\text{render\_tree})}; \\
\text{computePos(\text{render\_tree})}; \\
\]

\[
\text{computeHeight_\_computeWidth_\_computePos(\text{render\_tree})};
\]
What is Grafter?

Sound, Fine-Grained Traversal Fusion for Heterogeneous Trees (PLDI 2019)

https://dl.acm.org/citation.cfm?id=3314221.3314626
https://www.youtube.com/watch?v=j2henSFTZds
https://github.com/laithsakka/Grafter
This talk is not about Grafter! It's about utilizing Clang to implement Grafter

1. Embedded DSL in C++
2. Static analysis
3. Code generation
Why embedded DSL in C++

• Better Productivity.

• Better Performance.

• Ease of Integration.
Embedded DSL in C++

- Annotate components
- Verify annotated components against a set of rules.

```cpp
class __tree_structure__ ValueNode : public Node {
public:
    int Value;
    __tree_child__ Node *Left, *Right;
    __tree_traversal__ void search(int Key, bool ValidCall) override {
        Found = false;
        if (Key == Value) {
            Found = true;
            return;
        }
        Left->search(Key, Key < Value);
        Right->search(Key, Key >= Value);
        Found = Left->Found || Right->Found;
    }
    this->Left = this->Right;
    this->Left = new ValueNode();
    if (...) {
        Left->insert(...);
        Left->insert(...);
    }
}
```
Static analysis

1. A function is represented as a sequence of Clang::Stmt* and Clang::CallExpr* nodes. Different schedules are achieved by reordering statements and collapsing calls.

2. Understand accesses of statements and build call graphs to analyze dependences.
Code Generation

- Grafter builds the fused functions incrementally following a set of rewrite rules while tracking the original source code.

```cpp
void _fuse__F3F4(TextBox *r, int active_flags) {
    TextBox *r_f0 = (TextBox *)r;
    TextBox *r_f1 = (TextBox *)r;
    if (active_flags & 0b11) /*call*/ {
        unsigned int call_flags = 0;
        call_flags <<= 1;
        call_flags |= (0b01 & (active_flags >> 1));
        call_flags <<= 1;
        call_flags |= (0b01 & (active_flags >> 0));
        _r_f0->Next->_stub1(call_flags);
    }
    if (active_flags & 0b1) {
        _r_f0->Width = _r_f0->Text.Length;
        _r_f0->TotalWidth = _r_f0->Next->Width + _r_f0
                           ->Width;
    }
    if (active_flags & 0b10) {
        _r_f1->Height = _r_f1->Text.Length * (_r_f1->Width / CHAR_WIDTH) + 1;
        _r_f1->MaxHeight = _r_f1->Height;
        if (_r_f1->Next->Height > _r_f1->Height) {
            _r_f1->MaxHeight = _r_f1->Next->Height;
        }
    }
};
```
Code generation example

```c
void _fuse__F3F4(TextBox *r, int active_flags) {
    TextBox *r_f0 = *(TextBox *)(r);
    TextBox *r_f1 = *(TextBox *)(r);
    if (active_flags & 0b11) /*call*/ {
        unsigned int call_flags = 0;
        call_flags <<= 1;
        call_flags |= (0b01 & (active_flags >> 1));
        call_flags <<= 1;
        call_flags |= (0b01 & (active_flags >> 0));
        r_f0->Next->_stub1(call_flags);
    }
    if (active_flags & 0b1) {
        r_f0->Width = r_f0->Text.Length;
        r_f0->TotalWidth = r_f0->Next->Width + r_f0->Width;
    }
    if (active_flags & 0b10) {
        r_f1->Height = r_f1->Text.Length * (r_f1->Width / CHAR_WIDTH) + 1;
        r_f1->MaxHeight = r_f1->Height;
        if (_r_f1->Next->Height > _r_f1->Height) {
            _r_f1->MaxHeight = _r_f1->Next->Height;
        }
    }
};
```
Code generation example

- Replace original calls with calls to fused functions, and create virtual switches functions as needed.

```
int main()
{
    Element *ElementsList = ...;
    ElementsList->computeWidth();
    ElementsList->computeHeight();
}
```

```cpp
void TextBox::__stub1(int active_flags) {
    _fuse__F3F4(this, active_flags);
}
void Group::__stub1(int active_flags) {
    _fuse__F5F6(this, active_flags);
}
void End::__stub1(int active_flags) {
    _fuse__F1F2(this, active_flags);
}
int main()
{
    Group *ElementsList;
    //ElementsList->computeWidth();
    //ElementsList->computeHeight();
    ElementsList->__stub1(0b11);
}
It does scale..

We run Grafter on programs with more than 50 functions to be fused and automatically generate programs thousands lines of code that achieve significant speedups.
Conclusions

• Clang is useful in performing domain specific source to source transformations

• Easy to implement an embedded DSL in C++

• Clang AST is useful in collecting source-level information needed for static analysis

• Clang AST makes it easy to track input program while generating the output program