Using Clang as An Alternative C/C++ Frontend of The ROSE Source-to-Source Compiler

LLVM Developer Meeting, Oct 6-8, 2020

Anjia Wang¹,², Pei-Hung Lin¹, Chunhua Liao¹, Yonghong Yan²
Outline

● Motivation
● ROSE Compiler
  ○ Clang and EDG Frontend
  ○ Code Example: Source-to-Source Transformation
● Clang and ROSE AST
● Tech Details
  ○ Code Example: Clang AST to ROSE AST
● Conclusion and Future work
ROSE Compiler

- An open source compiler infrastructure to build **source-to-source** program transformation and analysis tools
- A unified AST as its IR for input codes written in C/C++ and Fortran
- Sophisticated compiler analyses, transformations and optimizations are developed on top of the AST and encapsulated as simple function calls
Motivation

- Clang: limited source-to-source translation support
  - Clang AST is immutable
  - No unparser to convert Clang AST to compilable source code
- ROSE: limitations of its current C/C++ EDG frontend
  - EDG is proprietary
  - Written in C with macros
  - EDG does not support OpenMP
Using Clang as An Alternative C/C++ Frontend of The ROSE

- C/C++
- Fortran

Clang

EDG

OFP

Unified AST for C/C++/Fortran

ROSE

C/C++ Unparsing

Fortran Unparsing

C/C++

Fortran
## Comparing Clang and EDG Frontend

<table>
<thead>
<tr>
<th></th>
<th>Clang</th>
<th>EDG</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>License</strong></td>
<td>Open Source</td>
<td>Proprietary</td>
</tr>
<tr>
<td><strong>SLOC</strong></td>
<td>$1.67 \times 10^6$</td>
<td>$1.59 \times 10^6$</td>
</tr>
<tr>
<td><strong>IR</strong></td>
<td>Clang AST</td>
<td>EDG IR</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>Modern C++11/14</td>
<td>C with macros</td>
</tr>
<tr>
<td><strong>User Community</strong></td>
<td>Any companies or universities</td>
<td>Compiler/tool vendors</td>
</tr>
<tr>
<td><strong>OpenMP</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
void axpy(int a, int* x, int* y, int size) {
    #pragma omp target map(to: x[0:size], a)
    map(tofrom: y[0:size])
    #pragma omp parallel for
    for (int i = 0; i < size; i++)
        y[i] = y[i] + a * x[i];
}

__global__ void OUT__1__8216__(int a,int *dev_x,int *dev_y)
{
    int _dev_lower,_dev_upper, ... ,_dev_thread_num,_dev_thread_id = ...;
    XOMP_static_sched_init(0,size - 1,1,1, _dev_thread_num, _dev_thread_id,
                        _dev_loop_chunk_size,_dev_loop_sched_index,_dev_loop_stride);
    while(XOMP_static_sched_next(&_dev_loop_sched_index,size - 1,1,_dev_loop_stride,
                        _dev_loop_chunk_size,_dev_thread_num,_dev_thread_id,_dev_lower,_dev_upper))
        for (i = _dev_lower; i <= _dev_upper; i += 1)
            _dev_y[i - 0] = _dev_y[i - 0] + a * _dev_x[i - 0];
}

void axpy(int a,int *x,int *y,int size)
{
    ...
    // transfer data and launch CUDA kernel
    int _threads_per_block_ = xomp_get_maxThreadsPerBlock(0);
    int _num_blocks_ = xomp_get_max1DBlock(0,size - 1 - 0 + 1);
    OUT__1__8216__<<<_num_blocks_,_threads_per_block>>>(a,_dev_x,_dev_y);
    xomp_deviceDataEnvironmentExit(0);
}
## Comparing Clang and ROSE AST

<table>
<thead>
<tr>
<th>Feature</th>
<th>Clang</th>
<th>ROSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutable</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Source-to-Source</td>
<td>Limited</td>
<td>Yes</td>
</tr>
<tr>
<td>Programming Language</td>
<td>C++11/14</td>
<td>C++</td>
</tr>
<tr>
<td>Represented Languages</td>
<td>C/C++</td>
<td>C/C++, Fortran</td>
</tr>
<tr>
<td>Unparsing</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>API</td>
<td>Create/Traverse</td>
<td>Create/Update/Delete/Traverse</td>
</tr>
</tbody>
</table>
Immutable Clang AST VS mutable ROSE AST

- **Clang immutable AST:**
  - Canonicalization of the “meaning” of nodes is possible once node is created
  - AST nodes can be reused when they have the same meaning
  - Serialization and deserialization support

- **ROSE mutable AST:**
  - Easily adding, deleting, and changing AST nodes from AST tree
  - With an elegant means of manipulating source code
  - Use with caution to avoid incorrect source location information, invalidated semantic information, and generating illegal program
Technical Details

- Clang AST Generation
  - Clang takes the C/C++ source code and creates an AST.

- Connector in ROSE
  - The connector in ROSE traverses the Clang AST and creates a ROSE AST accordingly.
Driver in ROSE for Converting Clang AST

- Creating an ASTConsumer and define conversation APIs for all Clang AST nodes
  - class ClangToSageTranslator : public clang::ASTConsumer {
    - virtual bool VisitDecl(clang::Decl * decl, SgNode ** node);
    - virtual bool VisitStmt(clang::Stmt * stmt, SgNode ** node);
    - virtual bool VisitType(clang::Type * type, SgNode ** node);
    - ...
  }

- Translation process:
  - Create compiler instance
    - clang::CompilerInstance
  - Inform the diagnostic client the beginning of source file processing
    - compiler_instance->getDiagnosticClient().BeginSourceFile(compiler_instance->getLangOpts(),
      &compiler_instance->getPreprocessor());
  - Parse specified file and notify AST consumer, translator, as the file is parsed.
    - clang::ParseAST(compiler_instance->getPreprocessor(), &translator, compiler_instance->getASTContext());
  - Inform the diagnostic client the ending of source file processing
    - compiler_instance->getDiagnosticClient().EndSourceFile();
Current Status: Supported Clang AST Node Types

Based on Clang 9 and excluding Objective-C support:

<table>
<thead>
<tr>
<th></th>
<th>Supported</th>
<th>Total</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>41</td>
<td>84</td>
<td>48.81%</td>
</tr>
<tr>
<td>Statement</td>
<td>63</td>
<td>198</td>
<td>31.82%</td>
</tr>
<tr>
<td>Type</td>
<td>18</td>
<td>58</td>
<td>31.03%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>122</strong></td>
<td><strong>340</strong></td>
<td><strong>35.88%</strong></td>
</tr>
</tbody>
</table>
Updates from Clang 9 to Clang 10

- **API changes**
  - ArrayRef'ized CompilerInvocation::CreateFromArgs
  - OpenMP token definitions moved from Clang into LLVM

- **Increased OpenMP 5.x support**
  - OpenMP master taskloop directive
  - OpenMP parallel master taskloop directive
  - OpenMP master taskloop simd directive
  - OpenMP parallel master taskloop simd directive
  - OpenMP parallel master directive
Clang AST and ROSE AST

```c
int calc(int a, int x, int y) {
    y = y + a * x + 10;
    return y;
}
```
bool ClangToSageTranslator::VisitFunctionDecl(clang::FunctionDecl * function_decl, SgNode ** node) {
    SgName name(function_decl->getNameAsString());
    SgType * ret_type = SageBuilder::buildTypeFromQualifiedType(function_decl->getReturnType());
    SgFunctionParameterList * param_list = SageBuilder::buildFunctionParameterList_nfi();
    applySourceRange(param_list, function_decl->getSourceRange());
    for (unsigned i = 0; i < function_decl->getNumParams(); i++) {
        SgNode * tmp_init_name = Traverse(function_decl->getParamDecl(i));
        SgInitializedName * init_name = isSgInitializedName(tmp_init_name);
        param_list->append_arg(init_name);
    }
    SgFunctionDeclaration * sg_function_decl = SageBuilder::buildNondefiningFunctionDeclaration(name, ret_type, param_list, NULL);
    SgInitializedNamePtrList & init_names = param_list->get_args();
    ... // rest of conversion
    *node = sg_function_decl;
    return VisitDeclaratorDecl(function_decl, node) && res;
}
Conclusion

● Clang works well with ROSE as an alternative C/C++ frontend.
  ○ Using Clang instead of EDG: open-source and better OpenMP support.
  ○ ROSE AST provides more flexible source-to-source transformation than Clang AST.

● Ongoing/future work
  ○ Upgrade Clang 9.x to Clang 10.x in ROSE.
  ○ Support the conversion of all the Clang AST nodes.
  ○ Replace OFP (Open Fortran Parser) with Flang.
Questions and Answers


This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.