CLAVA: C/C++ source-to-source from CMake using LARA

João Bispo, Pedro Pinto, João M. P. Cardoso
jbispo@fe.up.pt, p.pinto@fe.up.pt, jmpc@fe.up.pt

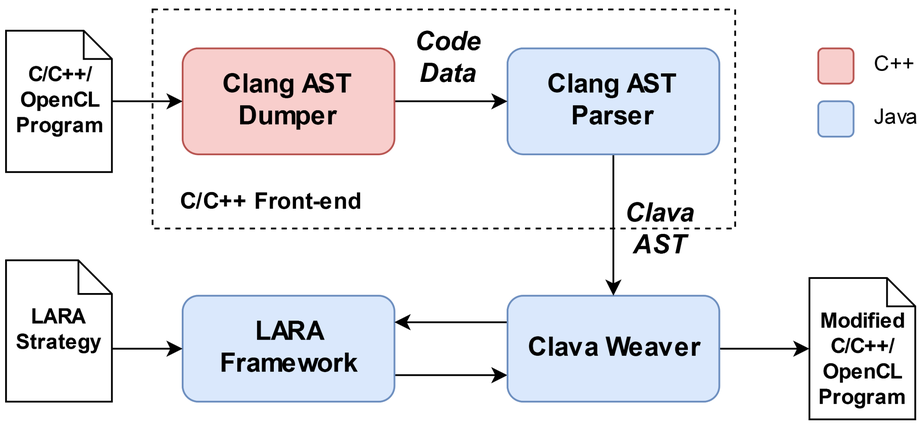
INESCTEC
U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

CLAVA

Introduction

- C/C++ source-to-source compiler written in Java
  - Built on top of the LARA framework
- Analysis and transformations written in LARA
  - DSL based on JavaScript for source code analysis and manipulation
- Custom AST for C/C++ code, based on the Clang AST
  - Uses Clang to parse code

CLAVA Toolflow and Clang Usage



- Initial idea: use Clang AST as IR for LARA
  - Prototype with ZeroMQ and Protocol Buffers
  - However, Clang source-to-source is text-based
- Development of CLAVA AST, in Java
  - AST is used to apply transformations and generate code
- Stand-alone C++ program (Clang AST Dumper)
  - Uses Clang as a library to dump the necessary data for CLAVA AST
  - Simplifies flow, one-way communication through output streams

Limitations

- Generated source-code has preprocessor elements resolved
  - Currently only reinserts include directives
- Does not cover all of C/C++
- CMake plugin is at proof-of-concept stage

Code Base

- CLAVA has ~36.000 SLoC-L
- Open-source, on GitHub

Component	SLoC-L	CLAVA %
LARA Framework	27395	N/A
Clang AST Dumper	3100	9%
Clang AST Parser	5372	15%
Clava AST	11719	33%
Clava Weaver	4927	14%
Generated	10946	30%

Execution Example

- Medium to large files
  - NAS benchmarks
  - Large single C programs
- Machine specs
  - Xeon E5-1650 3.60GHz
  - 64GB RAM
- LARA Code
  - Inserts code to create a dynamic call graph

Benchmark	Original SLoC-L	Inserted Lines	Parsing Time (s)	LARA Time (s)
NAS_EP	214	81	0.59	0.92
NAS_IS	243	63	0.58	0.95
NAS_CG	412	95	0.64	0.95
NAS_FT	523	112	0.70	0.91
NAS_MG	660	168	0.82	0.92
NAS_LU	1383	179	0.59	0.90
NAS_SP	1411	182	1.50	0.97
NAS_BT	1653	209	1.77	0.99
bzip2	3066	1125	1.52	1.38
gzip	3212	1106	0.96	1.37
oggenc	17462	4280	21.01	2.42

Selected Use Cases

- OpenMP Auto-parallelization
  - Static analysis of `for` loops
  - Uses Omega library for array analysis
  - Inserts OpenMP pragmas

```
select function{"foo"}.loop end
apply
    Parallelize.forLoops([$loop]);
end
```

- LAT
  - Design space exploration in LARA
  - Compiles, runs and collects results for multiple code versions

```
var x = new LatVarRange("x", 1, 10, 1);
var y = new LatVarRange("y", 1, 10, 1);
lat.setVariables([x, y]);
lat.tune();
```

- Code Generation for HDF5
  - File format/library to serialize values
  - Automatic generation of the boilerplate code

```
var recs = [];
select record end
apply recs.push($record); end
var hdf5Files = Hdf5.toLibrary(recs);
```

LARA + CMake Example

Clang & C++

```
//... includes


  bool VisitStmt(Stmt *s) {
    if (isa<IfStmt>(s)) {
      IfStmt *IfStatement = cast<IfStmt>(s);
      Stmt *Then = IfStatement->getThen();
      TheRewriter.InsertText(Then->getLocStart(),
        "// the 'if' part\n", true, true);
      Stmt *Else = IfStatement->getElse();
      if (Else)
        TheRewriter.InsertText(Else->getLocStart(),
           "// the 'else' part\n", true, true);
    }
    return true;
  }


  bool VisitFunctionDecl(FunctionDecl *f) {
    if (f->hasBody()) {
      Stmt *FuncBody = f->getBody();
      QualType QT = f->getReturnType();
      std::string TypeStr = QT.getAsString();
      DeclarationName DeclName = f->getNameInfo().getName();
      std::string FuncName = DeclName.getAsString();
      std::stringstream SSBefore;
      SSBefore << "// Begin function " << FuncName
        << " returning " << TypeStr << "\n";
      SourceLocation ST = f->getSourceRange().getBegin();
      TheRewriter.InsertText(ST, SSBefore.str(), true, true);
      std::stringstream SSAfter;
      SSAfter << "\n// End function " << FuncName;
      ST = FuncBody->getLocEnd().getLocWithOffset(1);
      TheRewriter.InsertText(ST, SSAfter.str(), true, true);
    }
    return true;
  }

//... setup and bootstrapping code
```

LARA

```
aspectdef LaraSample

  select if end
  apply
    $if.then.insert before "// the 'if' part";

    if($if.else !== undefined)
      $if.else.insert before "// the 'else' part";
  end

  select function{hasDefinition === true} end
  apply
    var typeStr = $function.functionType.returnType.code;
    var funcName = $function.name;

    $function.insert before "// Begin function "
      + funcName + " returning " + typeStr;
    $function.insert after "// End function " + funcName;
  end
end
```

CMake

```
cmake_minimum_required(VERSION 3.0)
project(lara_example CXX)

# Define program as usual in CMake
add_executable(lara_example "${SOURCES}")

...

# CLAVA CMake integration
find_package(Clava REQUIRED)

# Execute LARA code as a custom build step
# Target sources will point to generated files
clava_weave(lara_example LaraSample.lara)
```

- Simple code insertion example
  - Clang & C++: 57 SLoC-L
  - LARA: 12 SLoC-L
  - CMake: 2 SLoC-L (`find_package` and `clava_weave`)
- Requirements
  - Stand-alone JAR does not require any installation
  - Installation script for Linux (updater, CMake)
  - Runs on Ubuntu, CentOS, Windows and MacOS
- Features
  - Documentation generator
  - Unit testing
  - Standard library
  - CMake integration
- Try CLAVA
  - github.com/specs-feup/clava
  - specs.fe.up.pt/tools/clava

Acknowledgments

João Bispo acknowledges the support provided by Fundação para a Ciência e a Tecnologia, Portugal, under Post-Doctoral grant SFRH/BPD/118211/2016.