Challenges and Opportunities in C/C++ Source-to-Source Compilation

João Bispo (jbispo@fe.up.pt)
Nuno Paulino (nuno.m.paulino@inesctec.pt)
Luís Miguel Sousa (lm.sousa@fe.up.pt)

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

DEI
DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA

Outline

> Motivation

> Challenges

> Opportunities

> Examples

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

U.PORTO

Compiler

> Translates code written in one language to another one

> Programming language to machine code/executable
- gcc, clang, etc.

> Optimizing compiler
- Not only translates, but also transforms the code

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

U.PORTO

Source-to-Source Compiler

> Translates a high-level programming language to another high-level programming language
  - Often, the same language! (e.g., C to C)

> Useful for instrumentation, static analysis, code generation…

> Commonly used in certain application areas
  - JavaScript transpilers (e.g. TypeScript)

Compiler Research

> Traditional compiler tools

> Established and mature approach

> Low-level IRs

- GIMPL, LLVM-IR

> Catalogue of existing transformations

Compiler Research - Challenges

> Some information might be lost (e.g. comments, high-level structures, loops)

> High-learning curve

> Impractical distribution

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Source-to-Source Compiler - Motivation

- Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

- High-learning curve

- Impractical distribution

- Keep up with new compiler versions

- Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

Just another, separate tool

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

Just another, separate tool

> Keep up with new compiler versions

Source-code as the interface

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

Just another, separate tool

> Keep up with new compiler versions

Source-code as the interface

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Compatible with any compiler that accepts the language

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Challenges in Source-to-Source Compilation

> Limited support for the input languages

> Integration with existing toolchains

> Unintended interactions with the compiler

> Limitations in source code as an IR

> Competing technologies

Challenges - Limited support for the input languages

> Common for S2S tools to implement custom parsers

Challenges - Limited support for the input languages

> Common for S2S tools to implement custom parsers

Work	Codebase	Parser	Transformations	Extension mechanism
Clang[1] [31]	C/C++	Clang	Text-based	Framework
ROSE[2] [44]	C/C++	EDG	IR-based	Framework
Insieme[3] [18]	C/C++	Clang	IR-based	Framework
Cetus[4] [5]	Java	Custom	IR-based	Framework
Artisan [51]	Python	Clang	IR-based	Interpreter (Python)
CIL[5] [37]	C/OCaml	Custom	IR-based	Interpreter (OCaml)
Mercurium[6] [6]	C/C++	Custom	IR-based	Framework (dynamically loaded plugins)
Coccinelle[7] [28]	C/OCaml	Custom	Text-based	Interpreter (DSL)
Clava[8] [9]	Java	Clang	IR-based	Interpreter (JavaScript)

[1]https://github.com/llvm/llvm-project/tree/main/clang [2]https://github.com/rose-compiler
[3]https://github.com/insieme [4]https://github.com/hkhetawat/Cetus [5]https://github.com/cil-project/cil
[6]https://github.com/bsc-pm/mcxx [7]https://gitlab.inria.fr/coccinelle [8]https://github.com/specs-feup/clava

Challenges - Limited support for the input languages

> Common for C/C++ S2S tools to implement custom parsers

> C and C++ are complex languages

- Still in active development
- Preprocessor/templates

> Usually only support a limited subset

- E.g. ANSI C

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Challenges - Integration with existing toolchains

> Current C/C++ toolchains do not expect a source-to-source step

> Increased difficulties in development, debugging

> Limited concept of Source Maps

Challenges - Unintended interactions with the compiler

> Sometimes unclear how changes at a high-level affect compiled code

> Example: replacing operations with library calls[1]

- Prevents compiler optimizations

> Example: polyhedral loop optimizations[2]

- Transformations interfere with SSA
- Applied before all other optimizations (e.g. inlining)

[1] Christophe Denis, Pablo De Oliveira Castro, and Eric Petit. Verificarlo: Checking floating point accuracy through monte carlo arithmetic. *arXiv preprint arXiv:1509.01347*, 2015.

[2] Michael Kruse and Tobias Grosser. Delicm: scalar dependence removal at zero memory cost. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 241–253, 2018.

Challenges - Limitations in source code as an IR

> C/C++ source-to-source compilers according to transformations

- Text-based
- IR-based

> Text-based: preserve original source-code

> IR-based: traditional compiler passes, but over C/C++

> Parsimony in low-level IRs help analysis and transformations

- C and C++ much more complex in comparison

Challenges - Limitations in source code as an IR

Work		Codebase	Parser	Transformations	Extension mechanism
Clang[1]	[31]	C/C++	Clang	Text-based	Framework
ROSE[2]	[44]	C/C++	EDG	IR-based	Framework
Insieme[3]	[18]	C/C++	Clang	IR-based	Framework
Cetus[4]	[5]	Java	Custom	IR-based	Framework
Artisan	[51]	Python	Clang	IR-based	Interpreter (Python)
CIL[5]	[37]	C/OCaml	Custom	IR-based	Interpreter (OCaml)
Mercurium[6]	[6]	C/C++	Custom	IR-based	Framework (dynamically loaded plugins)
Coccinelle[7]	[28]	C/OCaml	Custom	Text-based	Interpreter (DSL)
Clava[8]	[9]	Java	Clang	IR-based	Interpreter (JavaScript)

[1] https://github.com/llvm/llvm-project/tree/main/clang [2] https://github.com/rose-compiler
[3] https://github.com/insieme [4] https://github.com/hkhetawat/Cetus [5] https://github.com/cil-project/cil
[6] https://github.com/bsc-pm/mcxx [7] https://gitlab.inria.fr/coccinelle [8] https://github.com/specs-feup/clava

Challenges - Competing technologies

> MLIR

- Addresses some LLVM-IR shortcomings
- Framework for building compilers

> Strong focus on:

- Moving between abstraction levels within the same IR
- Reusing compiler passes

> MLIR as a source-to-source competitor

- Preferential direction is lowering, but recent works address raising
- Starting point mainly DSLs, but increased interest in C/C++

Opportunities in Source-to-Source Compilation

> Reuse of existing parsers as-is

> Improved composability and compatibility

> Widening the scope and taming complexity

> Testing and Prototyping Environments

> Make compilers in general more accessible

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Opportunities - Reuse of existing parsers as-is

> Parsing C and C++ should be offloaded to 3rd party libraries

- Preferably without modifications
- Seriously consider if a custom parser is needed

> Examples

- EDG (ROSE)
- Clang (Insieme, Artisan, Clava)

Opportunities - Improved composability and compatibility

> Source language as the interface

- Natural integration in current C and C++ toolchains
- Source-to-source compilers can be interchangeable and mixed together

> Compiler as an interpreter, instead of a framework

- Extensions based on external APIs
- Improved distribuiton and reuse
- Lower entry barrier

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Opportunities - Improved composability and compatibility

Work		Codebase	Parser	Transformations	Extension mechanism
Clang[1]	[31]	C/C++	Clang	Text-based	Framework
ROSE[2]	[44]	C/C++	EDG	IR-based	Framework
Insieme[3]	[18]	C/C++	Clang	IR-based	Framework
Cetus[4]	[5]	Java	Custom	IR-based	Framework
Artisan	[51]	Python	Clang	IR-based	Interpreter (Python)
CIL[5]	[37]	C/OCaml	Custom	IR-based	Interpreter (OCaml)
Mercurium[6]	[6]	C/C++	Custom	IR-based	Framework (dynamically loaded plugins)
Coccinelle[7]	[28]	C/OCaml	Custom	Text-based	Interpreter (DSL)
Clava[8]	[9]	Java	Clang	IR-based	Interpreter (JavaScript)

[1] https://github.com/llvm/llvm-project/tree/main/clang [2] https://github.com/rose-compiler
[3] https://github.com/insieme [4] https://github.com/hkhetawat/Cetus [5] https://github.com/cil-project/cil
[6] https://github.com/bsc-pm/mcxx [7] https://gitlab.inria.fr/coccinelle [8] https://github.com/specs-feup/clava

Opportunities - Widening the scope and taming complexity

> Human-level use cases

- What a human programmer would do
- E.g., loop transformations, array flattening

> Compiler-level use cases

- Sequence of compiler passes
- Code can go directly to compiler

> Not mutually exclusive

- Compiler-level to extract information, human-level to apply it

Opportunities - Testing and Prototyping Environments

> Testing environment

- Source-to-source usually the first step in the tool-chain
- Allows complete flows (parsing, compiling, execution) within same script
- Natural fit for Design-Space Exploration (DSE)

> Prototyping environment

- Assuming a mature environment and compiler-level transformations
- Prototype as source-to-source before traditional compiler implementation

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO

Opportunities - Make compilers in general more accessible

> Previous opportunities can be applied to traditional compiler approach

> MLIR

- Compiler framework (vs interpreter)
- Programmed in C++

> Interest in addressing this!

- Python bindings for MLIR manipulation
- MLIR as a framework for creating source-to-source compilers?

Examples

> AutoPar - Automatic Parallelisation of for Loops

> Inline Assembly Insertion - RISC-V Custom Extensions

AutoPar - Automatic Parallelisation of for Loops

> AutoPar: Clava library for auto parallelization of C code

- Statically analyses and inserts OpenMP pragmas
- Automatic, no user effort
- Developed by Hamid Arabnejad, SPeCS Lab post-doc researcher

```
for(int i = 0; i < numIter; i++) {
   a += i;
}
```

Language: C

AutoPar - Automatic Parallelisation of for Loops

> [AutoPar]: Clava library for auto parallelization of C code

- Statically analyses and inserts OpenMP pragmas
- Automatic, no user effort
- Developed by Hamid Arabnejad, SPeCS Lab post-doc researcher

```c
#pragma omp parallel for default(shared) firstprivate(numIter) reduction(+ : a)
for(int i = 0; i < numIter; i++) {
   a += i;
}
```

Language: C

AutoPar - Automatic Parallelisation of for Loops

> Experiments:

- NAS – 2×
- PolyBench – 8.6× (8 threads, XL)
- Himeno – 10× (16 threads)

```c
#pragma omp parallel for default(shared) firstprivate(numIter) reduction(+ : a)
for(int i = 0; i < numIter; i++) {
   a += i;
}
```

Language: C

Prototyping RISC-V Custom Instructions

> RISC-V Processor

- Open standard ISA
- Designed to support customization

> Testing custom instructions

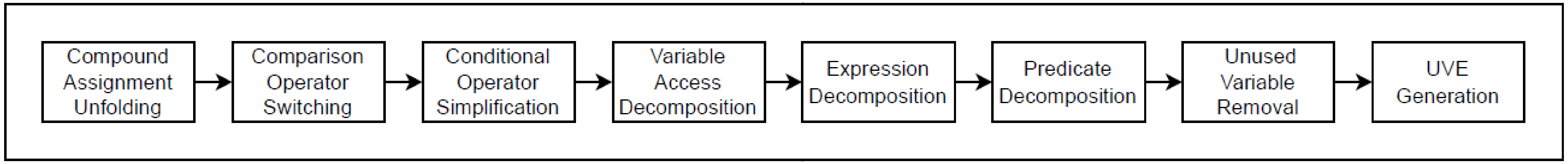
- Either manual insertion of assembly in C code (i.e. asm)
- Or add support in a traditional compiler toolchain (e.g. gcc, clang)
- Minimum: assembler that supports custom instructions

> Intermediate approach

- Automatically insert assembly in C code using source-to-source compilation

Prototyping RISC-V Custom Instructions

> Progressive lowering of C code, until UVE code generation



> Example: Compound Assignment Unfolding

```c
for (int i = 0; SIZE > i; i++) {
  dest[i] += src[i] * A;
}
```

```c
for (int i = 0; SIZE > i; i++) {
  dest[i] = dest[i] + src[i] * A;
}
```

Prototyping RISC-V Custom Instructions

```
void saxpy(size_t n, const float A, const float *src, float *dest) {
    for (size_t i = 0; i < n; i++)
        dest[i] = A * src[i] + dest[i];
}
```

```
asm volatile(
   "ss.st.d u0, %[RENCWJ], %[LNWHEG], %[VLXSSS] \n\t"
   "ss.ld.d u1, %[RENCWJ], %[LNWHEG], %[VLXSSS] \n\t"
   "ss.ld.d u2, %[NOBGQU], %[LNWHEG], %[VLXSSS] \n\t"
   "so.v.dp.d u3, %[CEFWVY], p0 \n\t"
   ".uve_loop_0_0_%=:  \n\t"
   "so.a.mul.fp u4, u2, u3, p0 \n\t"
   "so.a.add.fp u0, u1, u4, p0 \n\t"
   "so.b.nc u0, .uve_loop_0_0_%= \n\t"
   :: [RENCWJ] "r" (dest), [LNWHEG] "r" (128), [VLXSSS] "r" (1), [NOBGQU] "r" (src)
      , [CEFWVY] "r" (A));
```

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

U.PORTO

The End

THANKS!

> Clava GitHub: https://github.com/specs-feup/clava
  - Has link to tutorial

> Clava web demo: https://specs.fe.up.pt/tools/clava/

> Contact: jbispo@fe.up.pt