Formatting Fortran in the ROSE Compiler

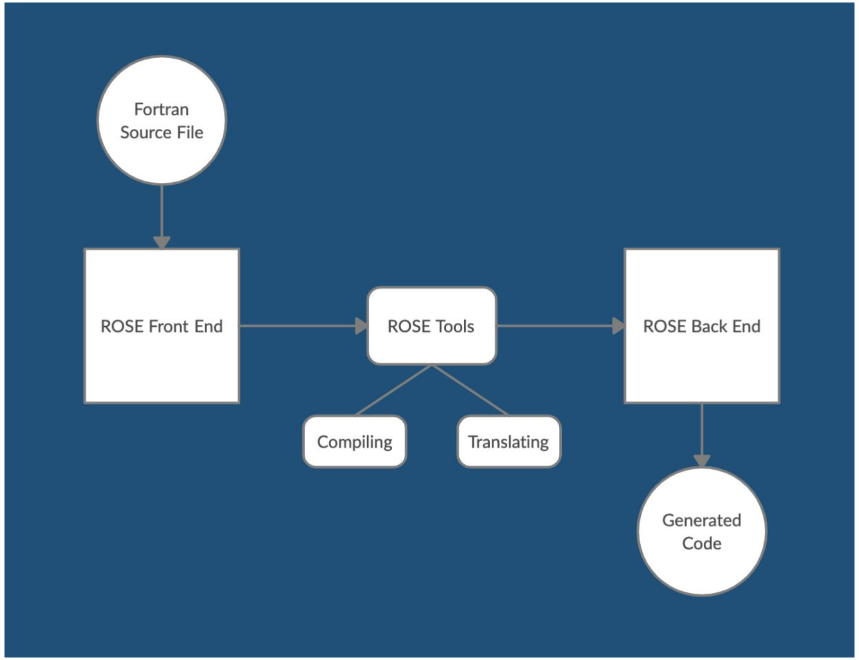
Skylar Neuendorff
SIParCS Program 2020
Rice University Class of 2021

NCAR
UCAR

Introduction

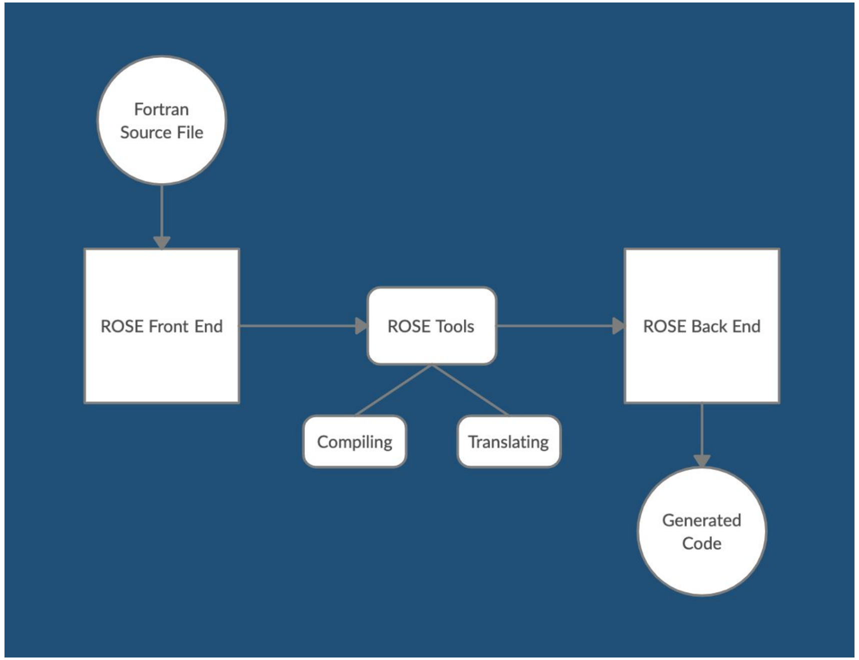
The ROSE Compiler

- The ROSE Compiler is a source-to-source translator
- Has resources for tools that can translate, analyze, and parse through C, C++, and Fortran files.



The ROSE Compiler

- Problem: The Fortran translator/compiler in ROSE loses a lot of the original formatting of the file.
- Goal: Work towards building a Fortran-Aware backend that preserves formatting.

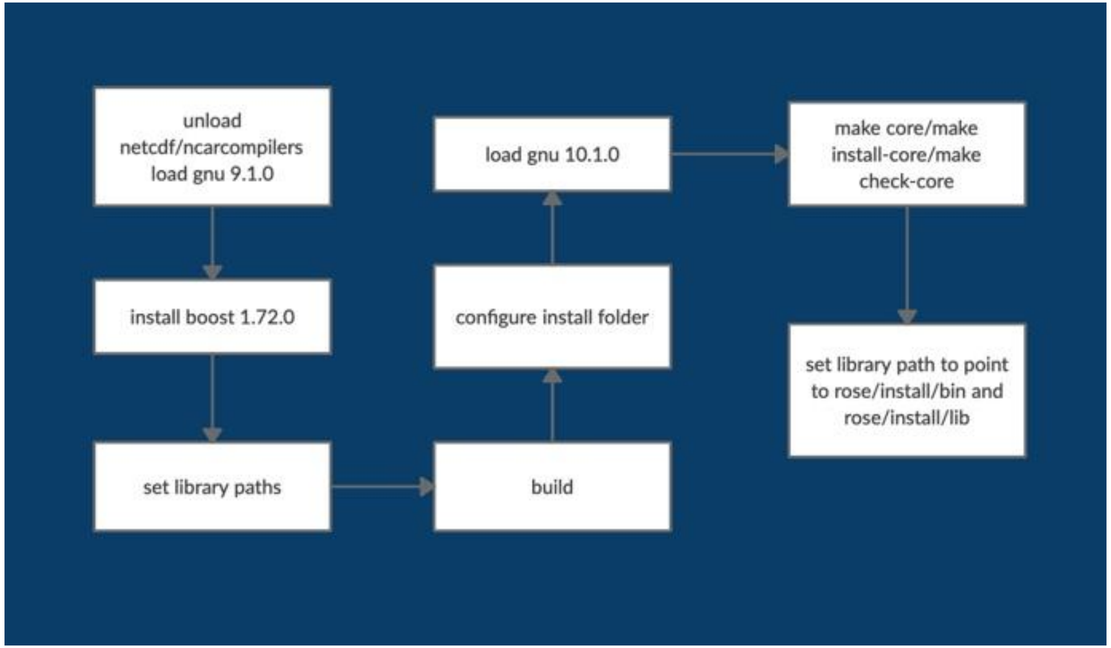


Building ROSE

Obstacles

- Biggest obstacle: ROSE lacks clear, up-to-date documentation of the versions of its dependencies that it is compatible with (e.g. Boost, GNU)
- Often, errors caused by incompatible dependencies are vague and unclear, so it took a lot of time to resolve them.
- Inability to find libraries even though they existed on the machine.
- Also had to work out the correct paths that had to be added to the Library Path, which again, are poorly documented in ROSE.

Successful Build For ROSE


unload netcdf/ncarcompilers load gnu 9.1.0
install boost 1.72.0
set library paths
build
configure install folder
load gnu 10.1.0
make core/make install-core/make check-core
set library path to point to rose/install/bin and rose/install/lib

Build Instructions

```
cd rose
PREFIX=`pwd`
module unload netcdf
module unload ncarcompilers
module load gnu/9.1.0
export CXX=$(which $CXX)
BOOST_ROOT="/glade/u/home/skylarfn/rose/boost_1_72_0/1.72.0/ins
tall"
cd boost_1_72_0/
./bootstrap.sh --prefix="${BOOST_ROOT}"
--with-libraries=chrono,date_time,filesystem,iostreams,program_option
s,random,regex,serialization,system,thread,wave
./b2 -std=c++11 -sNO_BZIP2=1 install
*allows it to find the crt1.o and crti.o files*
LIBRARY_PATH=/usr/lib64/:$LIBRARY_PATH
LD_LIBRARY_PATH=/usr/lib64/:$LD_LIBRARY_PATH
export LIBRARY_PATH
export LD_LIBRARY_PATH
export
LD_LIBRARY_PATH="/usr/lib64/jvm/java-1.8.0-openjdk-1.8.0/lib:$L
D_LIBRARY_PATH"
export
LD_LIBRARY_PATH="/usr/lib64/jvm/java-1.8.0-openjdk-1.8.0/jre/lib/am
d64/server:$LD_LIBRARY_PATH"
export
LD_LIBRARY_PATH="${BOOST_ROOT}/lib:$LD_LIBRARY_PATH"
./build
Mkdir ../build
Cd ../build
../rose-release/configure --prefix="${PREFIX}/install"
--enable-languages=c,c++,fortran
--with-boost="/${BOOST_ROOT}"
module load gnu/10.1.0
Make core -j10
Make install-core -j10
Make check-core -j10
export ROSE_SRC=${PREFIX}/rose-release
export ROSE_INSTALL=${PREFIX}/install # example
/data/rose/install
export PATH=${ROSE_INSTALL}/bin:${PATH}
export
LD_LIBRARY_PATH=${ROSE_INSTALL}/lib:${LD_LIBRARY_PATH}
```

Analyzing the Fortran Compiler

Points of Interest

- Examine what Fortran loses when it is parsed through the compiler
- Through this we can find out what Fortran needs in order to make a good copy of the original file.

Compiling Fortran Files

EXAMPLE:

Original test2008_53.f90

```
! This is an error since the function is assigned an implicit type
! and the declaration of the return type using the older syntax does
! not reset the type of the function.
function foo2()
    integer foo2
    foo2 = 1
end function
```

Rose compiled test2008_53.f90

```
! This is an error since the function is assigned an implicit type
! and the declaration of the return type using the older syntax does
! not reset the type of the function.
 FUNCTION foo2()
INTEGER :: foo2
foo2 = 1
END  FUNCTION
```

The compiler loses the following elements of the original files:

- Line indentation/spaces within lines
- Line breaks within single line of code
- Smaller comment blocks that occur within chunks of code
- Additionally, some functions are replaced by mathematical symbols (i.e. the LT function is replaced by the < symbol)
- Adds double-colon syntax "::" to all variable declarations

Why is this happening?

- When unparsing, ROSE has the option to use a UnparseFormatHelp object, which can contain info on how to insert spacing/indentation to maintain formatting, but otherwise it defaults to a standardized format (not usually consistent with the original file formatting)

- Currently, when unparsing fortran files, the UnparseFormatHelp object is null, so there is no info being passed along to help with formatting

Rewrites

- While I didn’t have time to handle all the formatting inconsistencies, I am rewriting unparse_format.C so that it preserves indentation.
- Each line has an Sg_File_Info object associated to it.
- Sg_File_Info contains the number of characters on a line, and the associated line number.
- unparse_format.C doesn’t have a copy of the original line of code.
- Rewrite: pass the original line of code to unparse_format.C to unparse_format.C to compare with the parsed version.
- Match formatting by comparison.

Conclusion/What’s Next?

What Comes After

- Next Steps: solve the remaining formatting errors in the Fortran Compiler. The most pressing issue is probably the occasional loss of comments, since these are often integral to understand the meaning of the code.
- My work provides a baseline for formatting and parsing through Fortran files and creating a usable Fortran-aware backend. Additionally, this work can be applied to other Fortran-based tools, like Fortran to C translation.

Thank You!

Thanks to the SIParCS program, and to my mentors Dan Nagle and Davide del Vento for all the help and support over the summer.