Hewlett Packard Enterprise

CHAPEL 1.25 RELEASE NOTES: COMPILER AND TOOL IMPROVEMENTS

Chapel Team

September 23, 2021

OUTLINE

LLVM BACK-END BY DEFAULT

LLVM BY DEFAULT
Background

Background:

- The compiler has traditionally generated C code to produce its executables
  - Requires a C compiler to build the final binary

- Has also had the ability to generate LLVM IR for many releases
  - Skips the step of generating and compiling C source files
  - Generates and compiles LLVM IR in memory

“The LLVM Core libraries provide a modern source- and target-independent optimizer, along with code generation support for many popular CPUs (as well as some less common ones!) These libraries are built around a well specified code representation known as the LLVM intermediate representation (‘LLVM IR’).”

- llvm.org

LLVM BY DEFAULT

This Effort: Overview

This Effort:

- Made LLVM the default back-end for Chapel
  - More opportunities for optimization vs. the C back-end
  - Promotes community involvement in developing the back-end by leveraging a common infrastructure
  - Decreases longer-term testing burden

- Fixed bugs in the compiler's LLVM back-end
  - Generating incorrect instructions
  - Mishandling signedness

LLVM BY DEFAULT

This Effort: Choosing between LLVM options and C

- Changed the default value of the CHPL_LLVM setting, as follows:

```
CHPL_LLVM=bundled    # if the bundled LLVM has already been built
CHPL_LLVM=system     # if a system LLVM installation is detected
CHPL_LLVM=none       # on systems where LLVM doesn't currently work for us, like linux32
CHPL_LLVM=unset      # otherwise
```

- Issue an error when building the compiler if CHPL_LLVM is detected to be 'unset'

```
Error: Please set the environment variable CHPL_LLVM to a supported value.
Supported values are:
  1) 'none' to build without LLVM support
  2) 'bundled' to build with the LLVM packaged in the third-party directory
  3) 'system' to use a pre-installed system-wide LLVM
```

LLVM BY DEFAULT
This Effort: Opting out of LLVM

- In cases where LLVM is the default, request the C back-end via CHPL_TARGET_COMPILER

```
export CHPL_TARGET_COMPILER=gnu
export CHPL_TARGET_COMPILER=<supported compiler>
```

- Supported C compilers are listed in the Environment section of the online documentation

- To disable LLVM entirely

```
export CHPL_LLVM=none
```

LLVM BY DEFAULT
Status, Next Steps

Status:

- LLVM is now the default back-end in nearly all configurations

Next Steps:

- Address performance regressions
  - Some tests lost performance with LLVM vs. the C back-end
  - 'chpl --fast' occasionally takes longer to compile with LLVM vs. the C back-end
- Upgrade from LLVM-11 to LLVM-12
- Investigate opportunities to further improve optimization with the LLVM back-end

SPECIFYING COMPILERS

SPECIFYING COMPILERS

Background, This Effort

Background:

- Generating and compiling C code was the default, but one could request LLVM code generation with '--llvm'
- There was no way to indicate the C or C++ compilation command / path
  - e.g., when 'CHPL_TARGET_COMPILER=gnu' the compilation would always use 'gcc'
- There was confusion about how 'CHPL_TARGET_COMPILER' interacts with the choice of C or LLVM strategies

This Effort:

- Deprecated '--llvm' and '--no-llvm' flags
- Now, LLVM code generation is the default, but it can be toggled by changing the target compiler
  - 'CHPL_TARGET_COMPILER=llvm' or '--target-compiler=llvm' requests LLVM code generation
  - 'CHPL_TARGET_COMPILER=gnu' or '--target-compiler=gnu' requests generating C code & compiling it with 'gcc'
- Additionally, 'CC' and 'CXX' environment variables are now available to control the C compiler command
  - 'CHPL_HOST_CC' / 'CHPL_HOST_CXX' and 'CHPL_TARGET_CC' / CHPL_TARGET_CXX' are also available when needed

SPECIFYING COMPILERS
Impact, Next Steps

Impact:

- Now 'make' only needs to build one runtime in LLVM-enabled configurations
- Resolved confusion about 'CHPL_TARGET_COMPILER' with the LLVM code generation
- Enabled a common strategy for setting the compiler command with 'CC' and 'CXX'
  - including specifying the complete path to the compiler:

```
CC=/usr/local/opt/llvm@11/bin/clang \
CXX=/usr/local/opt/llvm@11/bin/clang++ \
chpl myprogram.chpl
```

  - or requesting a particular version:

```
CC=gcc-10 CXX=g++-10 chpl myprogram.chpl
```

Next Steps:

- Should setting 'CC' / 'CXX' request C code generation? (issue #18450)
  - Currently, it does
  - Probably surprising if these are broadly set on a system to request a preferred C compiler
- Run down some other challenges/ambiguities that are emerging in the new approach (e.g., issue #18530)

C2CHAPEL IMPROVEMENTS

C2CHAPEL IMPROVEMENTS

Background: 'c2chapel' is a tool that takes C header files and generates Chapel C-bindings

- Attempting to use c2chapel with Apache Arrow/Parquet led to the discovery of many issues

```
typedef struct {
  int memberVar;
} intStruct;
```

$\longleftrightarrow$

```
extern record intStruct {
  var memberVar : c_int;
}
```

This Effort: Extended c2chapel to work with GNU extensions and fix bugs

- Added a new `--gnu-extensions` flag to use a parser capable of handling GNU expressions
  - As a result of this, c2chapel now requires Python 3.7 instead of 3.6 (affects Chapel's whole virtual environment)
- Included Chapel C-interop modules by default to support additional C types
- Fixed support for C structs that don't have an explicit `typedef`

Impact: Enabled c2chapel to fully parse Apache Arrow library

- Saves significant development time when enabling C library support
- Many c2chapel-generated programs now compile out of the box

OTHER COMPILER AND TOOL IMPROVEMENTS

OTHER COMPILER AND TOOL IMPROVEMENTS

For a more complete list of compiler and tool changes and improvements in the 1.25 release, refer to the following sections in the CHANGES.md file:

- ‘Tool Improvements’
- ‘Compilation-Time / Generated Code Improvements’
- ‘Portability’
- ‘GPU Computing’
- ‘Compiler Improvements’
- ‘Compiler Flags’
- ‘Bug Fixes’
- ‘Bug Fixes for Tools’

THANK YOU

https://chapel-lang.org
@ChapelLanguage