Hewlett Packard
Enterprise

# CHAPEL 1.29.0/1.30.0 RELEASE NOTES: DYNO UPDATES

Chapel Team

December 15, 2022 / March 23, 2023

# DYNO UPDATES OUTLINE

# BACKGROUND AND GOALS

# COMPILER REWORK EFFORT

- *dyno* is an ongoing effort to address problems with the Chapel compiler
- Focused on improving:
  - Speed
  - Error messages
  - Compiler structure and program representation
  - Compiler development

- Recent work has focused on:
  - Replacing the early compilation passes with incremental versions, including an incremental resolver
  - Improving error messages
  - Demonstrating an early form of separate compilation

# COMPILER REWORK DELIVERABLES

## Incremental Compilation Front-end

- Only re-parse and do type resolution on files that were edited
  - Could result in reducing compilation time
- Will still have the whole-program optimization and code-generation back-end

## Separate Compilation

- Make most of the optimizations happen per-file
- Will need a linking step for optimizations like function inlining that span files
- Should result in significantly faster compilation times

## Dynamic Compilation and Evaluation

- Enable Chapel code snippets to be written and run interactively
  - e.g., in Jupyter notebooks

Throughout the effort, improve the learning curve and error messages

# PROGRESS SINCE 1.28

# SUMMARY OF PROGRESS TOWARDS 1.29 AND 1.30 GOALS

1. Getting the dyno scope resolver ready for use in production
   - Goal was to use it in production in 1.30
   - Just missed the 1.30 release; already enabled on ‘main’ for the 1.31 release cycle

2. Building more components of the dyno resolver to reach feature-completeness
   - Goal was to have initial implementations of remaining major components of the new resolver by 1.30
   - Goal met: have at least draft components for all major areas & much work remains

3. Designing and implementing file formats and commands for separate compilation
   - Goal was to implement uAST serialization/deserialization, draft file format, and prototype separate compilation commands
   - Goal met: precompiled-header style support is in 1.30

4. Begin to improve error messages
   - Goal was to migrate errors from parsing and dyno scope resolution to a more user-friendly format by 1.29
   - Goal met: as of 1.30, parsing, checks, and new scope resolver errors include more user-friendly variants

# SUMMARY OF PROGRESS TOWARDS NEW GOALS

- Building the compiler with CMake
  - A new goal since 1.28
  - Now complete
- Investigating a compiler driver approach
  - A new goal since 1.28
  - Created a prototype but it needs more work

# SCOPE RESOLUTION

# SCOPE RESOLUTION: BACKGROUND

- *Scope resolution* is the process of matching identifiers with declared symbols
  - For example, in the following code, the 'arg' being printed refers the 'arg: string' formal

```
proc printArg(arg: string) {
  writeln(arg);
}
```

- Scope resolution is the next pass in the production compiler that we want to move to dyno
  - Involves re-implementing it in the new incremental framework

- In 1.28:
  - a '--dyno' flag was added to use dyno for scope resolution
  - the dyno scope resolver could handle 13,626 out of 14,020 test cases (97%)
    - caveat: only running the new scope resolver on user code (in order to enable incremental progress)
  - new scope resolver leaned on old compiler for some unhandled cases

# SCOPE RESOLUTION: THIS EFFORT

Further improved the dyno scope resolver to make it production-ready

- Added support for missing features
  - ‘private’ keyword, shadow scopes, record/class fields, …
- Enabled scope-resolving the built-in modules
- Still leaning on production scope resolver to handle gaps in implementation
- All tests pass (14518/14518)

# SCOPE RESOLUTION: IMPACT AND NEXT STEPS

**Impact:**

- Dyno scope resolver received numerous improvements, is now the default on ‘main’ (post-1.30)
- Users will see better error messages for errors reported during scope resolution
  - errors can now describe where a symbol came from
  - implemented improved errors for redefinition, unknown variables and modules, …
- Found bugs in production scope resolution and gaps in the language specification

**Next Steps:**

- Identify and fix gaps in the dyno scope-resolver that are currently handled by the production scope resolver
- Disable the production scope resolver in favor of the dyno scope resolver

# TYPE AND CALL RESOLUTION

# RESOLVING TYPES AND CALLS: BACKGROUND

- *Resolving* includes resolving types and resolving calls

```
var x = "hello";    // resolving a type: determine that 'x' has the type 'string'
f(1);               // resolving a call: determine that 'f(1)' calls 'f' below
proc f(arg: int) {}
```

- Resolution implements a large part of Chapel's semantics
  - It is also one of the major bottlenecks in the production compiler

- A new incremental resolver is part of the dyno effort

- Approach: get a draft of each major component in order to
  1. Raise language design issues before language stabilization
  2. Demonstrate integration of all resolver components in the new resolver effort

# RESOLVING TYPES AND CALLS: STATUS

- Currently have draft implementations for handling the following features:
  - progress since Sept 2022 in **bold** — many of these need more work

- generic instantiation
- param folding
- implicit conversions
- tuple types
- type construction
- varargs functions
- loop index variables
- param loops
- enums
- method calls
- function disambiguation
- 'new R( )' runs 'R.init( )'

- '?t' in formals
- caching of instantiations
- compiler-generated functions
- fields
- parenless methods
- split init
- copy elision
- task/loop intents

- **initializer bodies**
- **split init and copy elision**
- **operator overloads**

- **reductions**
- **task/loop intents**
- **const checking**
- **'forwarding'**
- **return intent overloading**
- **ref-if-modified for e.g. arrays**
- **generating calls e.g. 'deinit'**
- **error types for 'catch'**
- **arrays & domains**
- **try / throws checking**
- **reflection**

# SEPARATE COMPILATION

# SEPARATE COMPILATION

**Background:** Separate compilation has been a long-desired feature for Chapel

**This Effort:** Added experimental support for libraries that store pre-parsed '.chpl' files

- Introduced serialization in frontend for uAST and some internal types
  - uAST is dyno's internal representation of a program
- Stored in '.dyno' files which can be used on the command line with 'chpl'
  - Note: the '.dyno' filetype is an unstable prototype

```
$ chpl --dyno-gen-lib MyLibrary.chpl      # creates 'MyLibrary.dyno'
$ chpl MyApp.chpl MyLibrary.dyno          # skips parsing for "MyLibrary"
```

**Status:** Made a first step towards separate compilation

- Will build upon serialization infrastructure for future efforts

**Next Steps:**

- Choose user-facing flags and file extension
- Explore storing compiled concrete functions in library files
- Improve performance and space efficiency

# IMPROVED ERROR MESSAGES

# IMPROVED ERROR MESSAGES: THIS EFFORT

- Since 1.28, started implementing a new error message system
  - inspired by languages such as Elm, Rust
  - goal is to make clearer, more specific, better-looking error messages
  - implemented new error messages in the new components
    - as dyno's resolver takes over more functionality, more errors will be improved

- Error system features:
  - Detailed and brief verbosity modes aimed at experts and new users, respectively
  - Detailed mode includes code printing and underlining
  - Each error has a name (e.g. 'Deprecation', 'IncompatibleIfBranches')
    - Intended for use with silencing / escalation, linking documentation

# IMPROVED ERROR MESSAGES: EXAMPLE

program.chpl

```
1.  module Lib1 { }
2.  module Lib2 {
3.    var a: int;
4.  }
5.  module T2 {
6.    public use Lib1, Lib2;
7.    var a: real;
8.    proc main() {
9.      writeln(a);
10.   }
11. }
```

Error (production compiler):

```
program.chpl:7: error: symbol a is multiply defined
program.chpl:3: note: also defined here
```

Error (dyno, brief):

```
program.chpl:5: In module 'T2':
program.chpl:7: error: 'a' has multiple definitions
program.chpl:6: note: redefined through the 'use' statement here
program.chpl:3: note: leading to the definition here
```

Additional context

More information about the source of the conflict

Mentioning ‘a’ no longer required to trigger the error

# IMPROVED ERROR MESSAGES: EXAMPLE, DETAILED OUTPUT

program.chpl

```
1.  module Lib1 { }
2.  module Lib2 {
3.    var a: int;
4.  }
5.  module T2 {
6.    public use Lib1, Lib2;
7.    var a: real;
8.    proc main() {
9.      writeln(a);
10.   }
11. }
```

Error (dyno, detailed):

```
── error in program.chpl:7 [Redefinition] ──
  'a' has multiple definitions.
  It was first defined here:
      |
    7 |     var a: real;
      |         ^^^^^^^
      |
  Redefined through the 'use' statement here:
      |
    6 |     public use Lib1, Lib2;
      |                      ^^^^
      |
  Leading to the definition here:
      |
    3 |     var a: int;
      |         ^^^^^^
      |
      |
```

Includes name of error

Additional, plain English description

Underlined relevant part of ‘use’

# IMPROVED ERROR MESSAGES: STATUS, NEXT STEPS

**Status:**

- Parsing errors, control flow checks now use the new error message format
- Dyno scope resolver (off by default, enabled with '--dyno') also uses new format

**Next Steps:**

- Continue migrating error messages
- Continue to improve the detailed error format
- Resolve open questions around displaying errors that span multiple files, line breaking and wrapping

# BUILDING THE COMPILER WITH CMAKE

# BUILDING WITH CMAKE

**Background:** Chapel has historically relied on Makefiles to build compiler, runtime, and tool components

- However, LLVM requires CMake
- CMake is also the preferred way to build the new dyno frontend and is required to build it as a library
  - frontend was temporarily adjusted to also include Makefiles to support building 'chpl' and 'chpldoc'

**This Effort:** Use CMake behind-the-scenes to build 'chpl' and 'chpldoc'

**Impact:** Improves developer QOL while maintaining compatibility with existing build scripts

- existing 'make' commands work as before
- new CMake targets integrate nicely with IDEs and provide enhanced lookup features for symbols
- faster parallel builds
- direct use of CMake allows developers to opt into generating Ninja build system files rather than Makefiles

**Next Steps:** Migrate additional components away from Makefiles as needed

# MIGRATING TO A COMPILER DRIVER

# COMPILER DRIVER

**Background:** The Chapel compiler process remains live during backend compilation

- System memory used by the earlier parts of Chapel compilation remains in use during backend C / LLVM linking
- Results in higher memory pressure than necessary

**This Effort:** Explored using a compiler driver with subprocesses for each phase of compilation

- This is common: both Clang and GCC use a compiler driver
- ‘chpl’ can act as the driver and provide outputs from one phase as inputs to another phase

**Status:** Prototyped on a feature branch

- Prototype can compile most code with C or LLVM backend

**Next Steps:** Resolve remaining bugs and test failures

- Improve prototype to production quality and add more testing
- Gather data on performance changes and consider changing to use driver mode by default

# DYNO GOALS FOR 1.31 AND 1.32

# SUMMARY OF DYNO'S GOALS FOR 1.31 AND 1.32

Dyno efforts will include these separable directions of effort, approximately in priority order

1. Frontend Integration & Improving New Resolver
   - Goal: able to disable production scope resolver by 1.31
   - Goal: new resolver works in opt-in mode for end-to-end compilation for most tests by 1.32
2. Separate Compilation
   - Goal: Demonstrate a concrete function saved in the library files by 1.32
3. Incremental Compilation
   - Goal: Demonstrate live scope resolution from an editor by 1.31
4. Compiler Driver
   - Goal: Get opt-in compiler driver support merged for 1.31

- While working on (1) above, taking care to help with language stabilization efforts
  - The dyno effort has been identifying language issues at a steady pace so far
  - Raising and addressing language features that seem poorly specified, not specified, or that are too confusing

# OTHER DYNO IMPROVEMENTS

# OTHER DYNO IMPROVEMENTS

For a more complete list of dyno changes and improvements in the 1.29.0 and 1.30.0 releases, refer to the following section in the CHANGES.md file:

- Developer-oriented changes: 'dyno' Compiler improvements/changes

# THANK YOU

https://chapel-lang.org
@ChapelLanguage