Compiler and Tool Changes

Chapel version 1.19

March 21, 2019


chapel_info@cray.com


chapel-lang.org


@ChapelLanguage

Outline

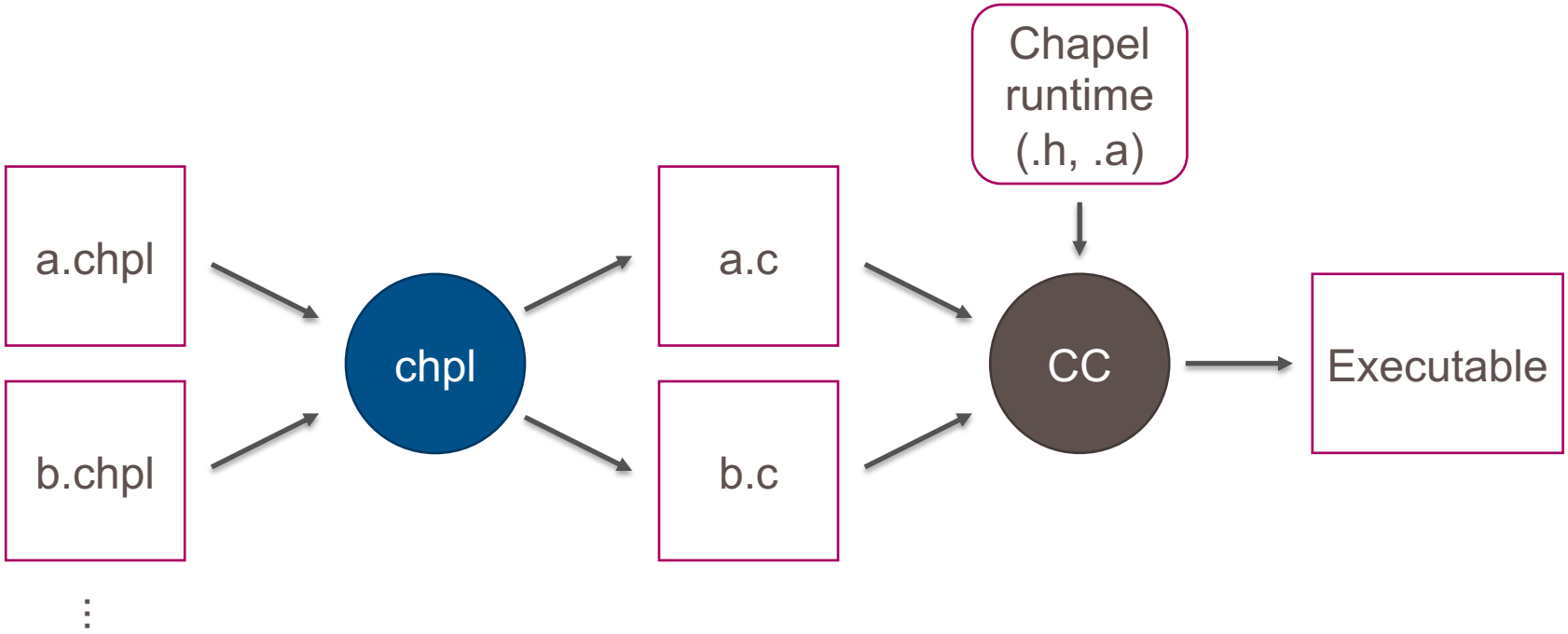
- LLVM Improvements
- Specifying Target Architecture
- Disambiguating Param Configs
- Retiring chpl-ipe

LLVM Improvements

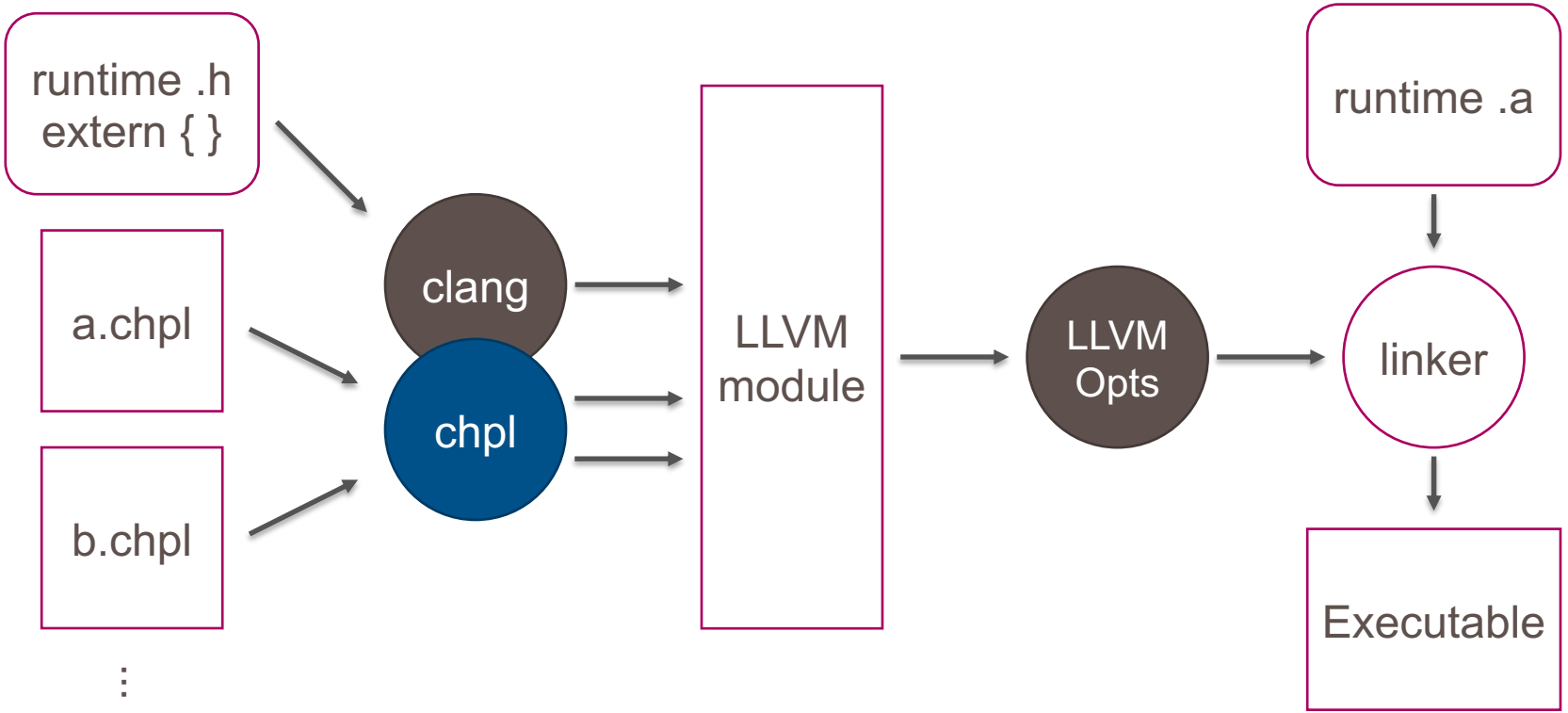
LLVM: Background

- LLVM is a compiler optimization framework
  - Actively developed and constantly improving
- The Chapel compiler generates C code by default
  - Runs a C compiler to compile the generated code
  - But can generate LLVM Intermediate Representation instead
- We want the Chapel compiler to use LLVM by default
  - To reduce maintenance vs. depending on many C compilers
  - To enable new optimization opportunities

Chapel --llvm compilation flow

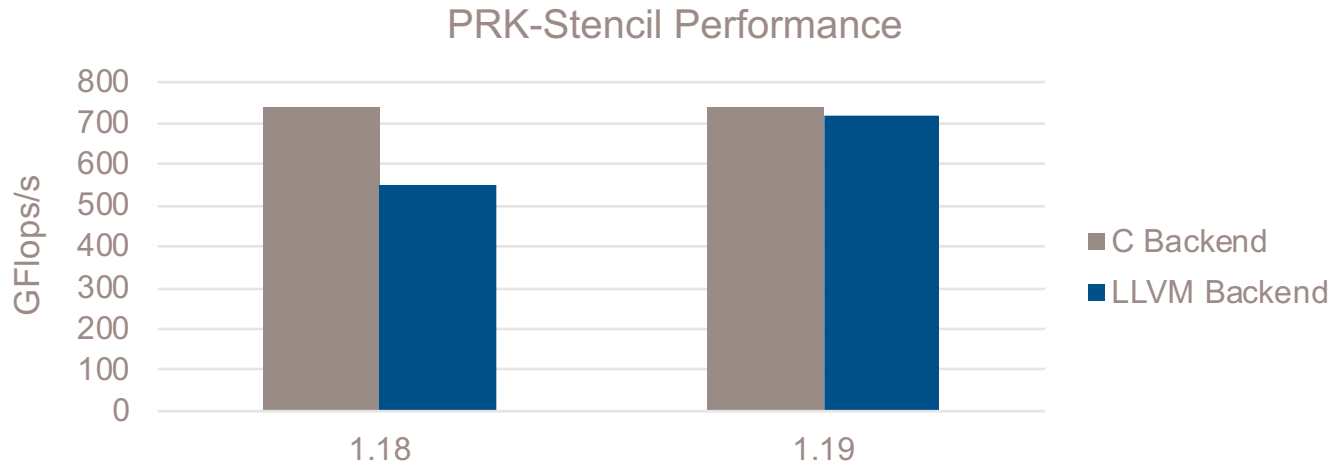


Chapel --llvm compilation flow



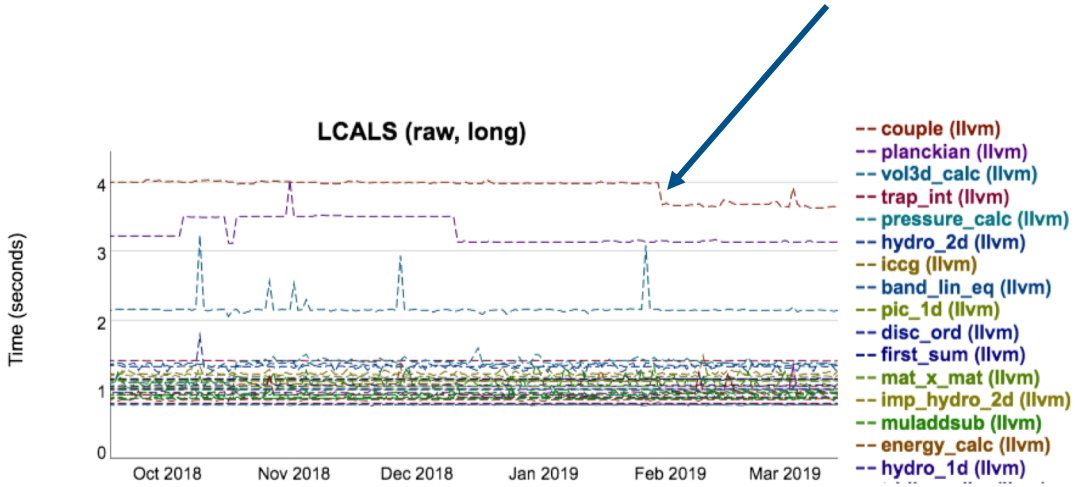
LLVM: PRK Stencil Improvement

- PRK stencil had poor multi-locale performance with --llvm
- Optimization was limited by alias analysis between C and Chapel types
- Migrated a privatization function from the C runtime to Chapel
  - Long-term, would like to improve alias analysis



LLVM: Upgrade to LLVM 7

- Upgraded the included LLVM from LLVM 6 to LLVM 7
  - Provided close to 10% improvement for LCALS 'couple' kernel

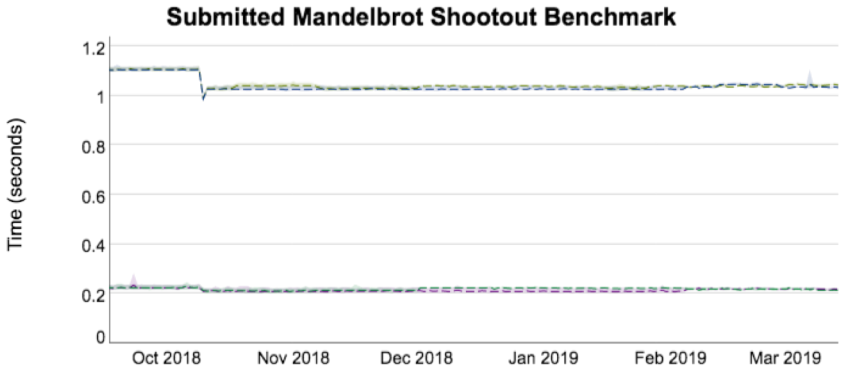


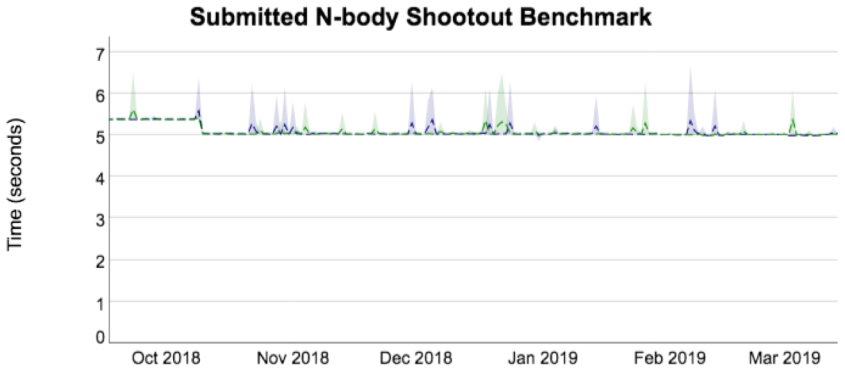
LLVM: Enhanced compatibility

- The LLVM back-end can now link statically
- Now compatible with LLVM versions 6-8
- Now compatible with MacOS Mojave
- Removed ‘register’ keyword from Qthreads (deprecated in C++)
  - Our change was merged upstream
  - Enabled '--llvm' to work on 32-bit x86
- Improved ability to find header files in more diverse installations
- Improved ability to invoke the optimizer on Cray XC systems

LLVM: Floating point optimization

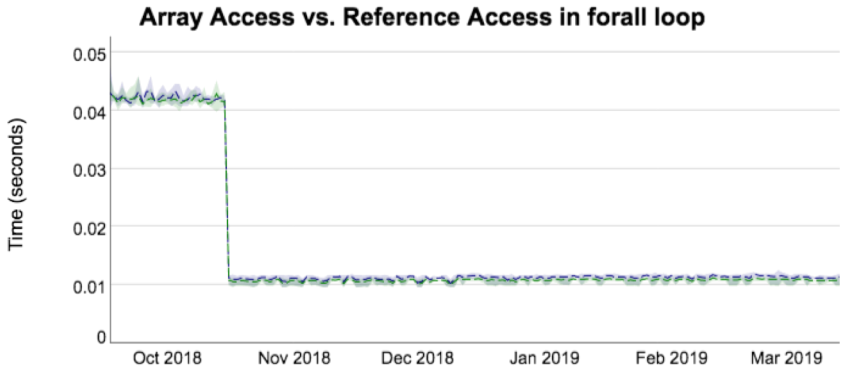
- Configured LLVM optimization to use fused floating-point multiply-add by default
- Resulted in a modest performance improvement for two benchmarks

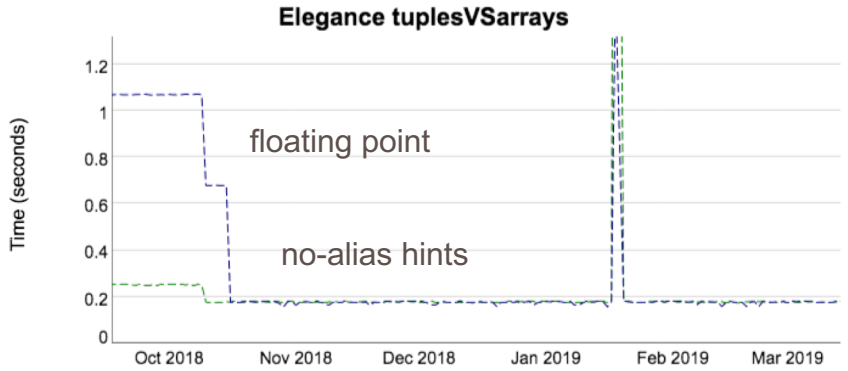




LLVM: no-alias hints

- Added new analysis pass to determine when Chapel arrays cannot alias
- Used the result to communicate to LLVM IR when array dereferences don't alias
  - via ‘noalias’ metadata
- Provided close to 2x performance improvement for some benchmarks





LLVM: Vectorization

- LLVM includes a parallel_loop_access loop metadata hint
  - Indicating to optimization no memory carried dependencies
- LLVM's LoopVectorizer is pretty good at adding runtime checks
  - To verify that aliasing doesn't introduce memory dependencies
- Now chpl emits parallel_loop_access metadata for forall and vectorizeOnly loops
  - But only when we are confident vectorization is safe
  - Can improve performance when the cost of the runtime check is significant
- Investigated prototype integration with the Region Vectorizer
  - A more general vectorizer than the LLVM Loop Vectorizer
  - Shows promising speedups for some benchmarks

LLVM: Next Steps

- Implement full ABI compatibility
- Make --llvm the default for the 1.20 release
- Study more benchmarks to try and improve performance
- Continue to work on Region Vectorizer integration
  - Mark more loops as vectorizeable
- Improve alias analysis between C and Chapel types

Choosing the Target Architecture and CPU for Specialization

target cpu: Background

- Before 1.19, there was no way to specify machine type
  - CHPL_TARGET_PLATFORM indicated an OS, e.g. linux64
  - CHPL_TARGET_ARCH specified a processor type, e.g. sandybridge
- In fact CHPL_TARGET_ARCH was used to determine 3 things:
  - A processor architecture, e.g. x86_64
  - A processor implementation, e.g sandybridge
  - A symbolic processor type for specialization/optimization, e.g. native
- Led to problems when one chapel directory is used for incompatible architectures
  - CHPL_TARGET_ARCH=native used same paths for both
  - Compiler build and bin directory did not differentiate at all

target cpu: This Effort

- Split CHPL_TARGET_ARCH into two flags
  - CHPL_TARGET_ARCH now means processor architecture
    - For example x86_64, arm, aarch64, ppc64
  - CHPL_TARGET_CPU now means CPU type for specialization
    - For example sandybridge, k8, native, none, sandybridge
- Added CHPL_HOST_ARCH, used it in compiler paths
  - Compiler is in bin/$CHPL_HOST_PLATFORM-$CHPL_HOST_ARCH
  - For example bin/linux64-x86_64/chpl
- Updated build and third-party install paths

target cpu: Impact

- Now possible to work with different architectures when
  - Building Chapel from source from a shared directory
  - Installing Chapel to a shared directory
  - Including when using CHPL_TARGET_CPU=native

Compile-time Config Disambiguation

Config Disambiguation: Background

Background:

- Distinct modules may happen to use the same config names

```
module M1 { config const debug = false; … }
module M2 { config const debug = true; … }
```

- Chapel has long supported execution-time disambiguation of configs:

```
$ ./myProg --M1.debug=true --M2.debug=false
```

- However, compile-time configs have not received similar support

```
$ chpl myProg.chpl -sdebug=true
myProg.chpl:2: error: Ambiguous config param or type name …
$ chpl myProg.chpl -sM1.debug=true
error: Trying to set unrecognized config param 'M1.debug' …
```

Config Disambiguation: This Effort and Impact

This Effort:

- Added disambiguation support for compile-time configs
- Improved error messages when ambiguity occurs

Impact: Compile-time disambiguation now works much better:

```
$ chpl myProg.chpl -sdebug=true
myProg.chpl:2: error: ambiguous config name (debug)
myProg.chpl:1: note: also defined here
myProg.chpl:2: note: (disambiguate using -s<modulename>.debug…)
$ chpl myProg.chpl -sM1.debug=true
$ ./myProg
```

Config Disambiguation: Next Steps

Next Steps:

- Extend disambiguation to support module paths
  - Currently only supports disambiguation of top-level module configs

```
module M1 {
  module M2 {
    config const debug = false;
  }
}
```

```
$ chpl -sM1.M2.debug=true myProg.chpl
$ ./myProg --M1.M2.debug=true
```


Retiring chpl-ipe

chpl-ipe: Background and This Effort

CRAY

Background:

- ‘chpl-ipe’ provided a prototype interactive Chapel environment
  - Added in Chapel 1.11.0
  - A proof-of-concept that supported minimal Chapel features
  - Took the approach of incrementally parsing code, interpreting the IR
  - Has received little attention or upkeep since then
- Main takeaway: interactive Chapel warrants a different compilation approach
- Meanwhile, we were carrying around the code, documentation, issues, …

This Effort: Removed chpl-ipe from the release to clear the slate

chpl-ipe: Status and Next Steps

Status:

- Still want to support Chapel in an interactive mode
- Have some proposals for how that could be achieved
  - Though they tend to require dramatic changes to the compiler
  - Such changes would likely help with other things too (e.g., compile time)

Next Steps:

- Determine the right timing and resources to pursue this
- Decide upon and prototype a new approach

For More Information

For a more complete list of compiler and tools changes in the 1.19 release, refer to 'Feature Improvements', 'Compiler Improvements', 'Performance Optimizations/Improvements', 'Portability', 'Packaging / Configuration Changes', 'Deprecated and Removed Features, and other sections in the CHANGES.md file.

SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

THANK YOU

QUESTIONS?


chapel_info@cray.com


@ChapelLanguage


chapel-lang.org


cray.com


@cray_inc


linkedin.com/company/cray-inc-