Compiler / Tools

Chapel Team, Cray Inc.

Chapel version 1.17

April 5, 2018

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.

Outline

- Mason Improvements
- Bash Tab Completion For chpl
- Compiler Flag Suggestions
- Default Executable Name Change
- LLVM Back-end Improvements
- Communication Optimization with --llvm-wide-opt
- Other Compiler/Tool Improvements

Mason Improvements

Mason: Background

- **Mason is the Chapel package manager**
  - Supports commands for completing different tasks

```
new       Create a new mason project
update    Update/Generate Mason.lock
build     Compile the current project
run       Build and execute src/<project name>.chpl
search    Search the registry for packages
env       Print environment variables recognized by mason
clean     Remove the target directory
doc       Build this project's documentation
```

  - Uses a registry containing “Bricks” describing packages
    - Default registry is publicly hosted at github.com/chapel-lang/mason-registry
    - MASON_REGISTRY environment variable overrides the default location
    - Only one registry can be used at a time

Mason: This Effort

- **Added support for multiple mason registries**
  - MASON_REGISTRY is now a comma separated list of registries
    - Each registry has an optional “name|” prefix to name a local directory to use
    - “name” defaults to the text following the final slash in the location

```
export MASON_REGISTRY="loc|/path/to/reg1,http://reg2.com/reg"
```

- **Added two new mason commands**

```
help      Display a help message
version   Display the mason version number
```

- **Added “make install” support**
  - After building mason it can be installed next to the “chpl” binary

Mason: Impact

- **Mason can now use multiple registries**
  - Bricks are searched for in left-to-right order of MASON_REGISTRY
  - Registries can be local and include local, private packages
  - Each registry can be named locally using MASON_REGISTRY
- **Mason can easily be installed next to the chpl binary**

```
make install
```

- **Show help/version messages with mason commands**

```
mason help
mason version
```

Mason: Next Steps

Next Steps: Continue to add and harden Mason features

- Add mason commands for additional functions

```
add       add a dependency
rm        remove a dependency
init      create a project in an existing directory
test      run project tests
...
```

- Add support for C dependencies
- Simplify creation of registries and adding new Bricks
- Improve error messages
- Add continuous integration testing for the package ecosystem

Bash Tab-completion for chpl

Tab-completion for chpl

Background: There are many verbosely-named chpl options

- Finding the right option requires searching help output or man pages

```
chpl --help
man chpl
```

This Effort: Add a bash tab-completion script for chpl

- Script knows about all compiler options and can autocomplete them
- For multiple matches, prints them and completes as much as possible

Impact: Bash users can autocomplete chpl options

- Bash users can use tab-completion for chpl compiler options

```
source $CHPL_HOME/util/chpl-completion.bash
```

Next steps: Developer vs. non-developer options

- Only autocomplete developer options when in developer mode

Compiler Flag Suggestions

Flag Suggestions

Background: Compiler gave a generic error for misspelled flags

```
$ chpl -fast
Unrecognized flag: '-f' (use '-h' for help)
$ chpl --ieee
Unrecognized flag: '--ieee' (use '-h' for help)
```

This Effort: Compiler suggests a flag in simple cases

```
$ chpl -fast
Unrecognized flag: '-f' (use '-h' for help)
       Did you mean --fast ?
$ chpl --ieee
Unrecognized flag: '--ieee' (use '-h' for help)
       Did you mean --ieee-float ?
```

Impact: Compiler is more friendly

Default Executable Name Change

Executable Name: Background

Background:

- Historically, compiling `foo.chpl` resulted in the executable `a.out`
- In 1.16, executable started being named after the main module
  - Why?
    - because every program has a single main module (vs. multiple files and modules)
    - because in practice the main module typically takes its name from its file

- However, this led to confusion in certain cases:

myProgram.chpl:

```
module M1 {
  writeln("Hello!");
}
```

```
> chpl myProgram.chpl
> ./myProgram
./myProgram: No such file or directory
```

- Users are accustomed to executables taking the name of *some* file

Executable Name: This Effort and Impact

This Effort:

- 1.17 names the executable after the file containing the main module
  - Why?
    - still uses something unique about the program
    - avoids the surprising cases that 1.16 had
    - returns to normal situation of naming executables after files
    - still supports the common case of the main module taking its name from its file

Impact:

myProgram.chpl:

```
module M1 {
  writeln("Hello!");
}
```

```
> chpl myProgram.chpl
> ./myProgram
Hello!
```

LLVM Back-end Improvements

LLVM Back-end Improvements: Background

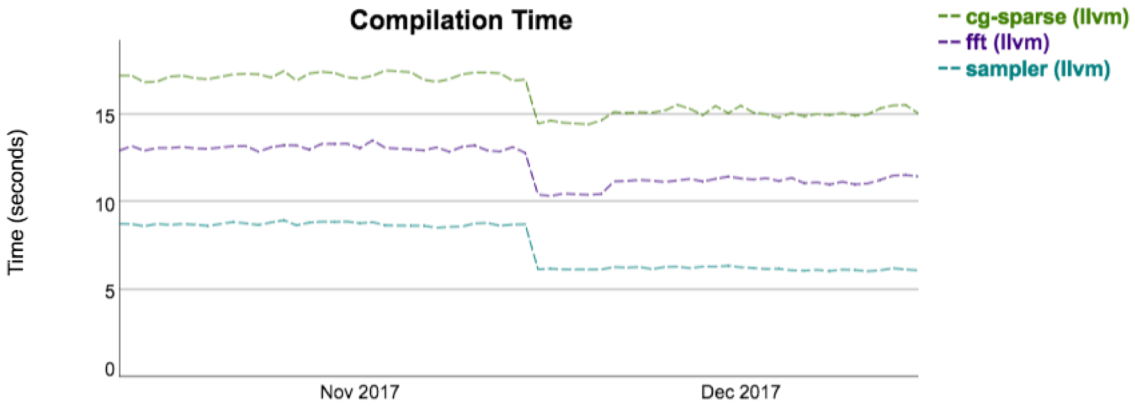
- **LLVM is a compiler optimization framework**
  - actively developed and constantly improving
- **We want LLVM to become our default back end**
  - to focus our attention instead of dividing it among C compilers
  - to improve optimization
  - to enable communication optimization

LLVM Back-end Improvements: This Effort

- Ported Chapel to LLVM 6.0
- Removed support for LLVM versions older than 4.0
- CHPL_LLVM=system now supports Mac Homebrew
- Improved precision of LLVM alias analysis metadata
- Improved --llvm compilation speed
- Addressed problems with --llvm-wide-opt
  - See: next section

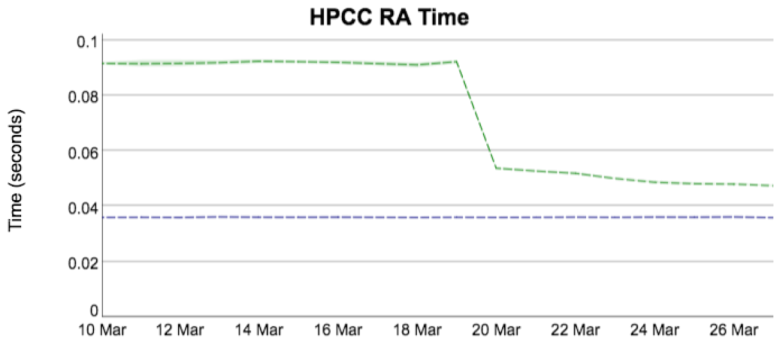
LLVM: Impact: Compilation Time

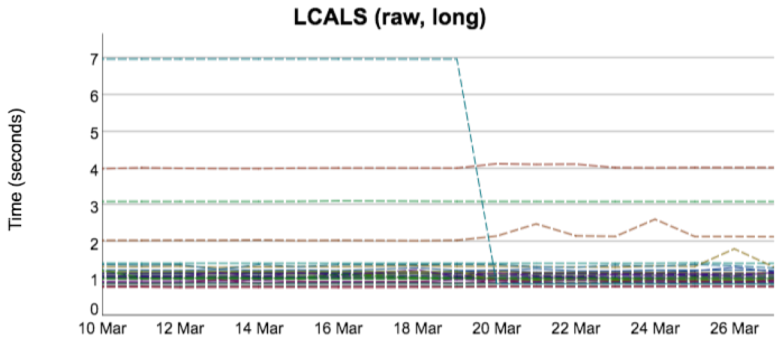
- **--llvm compilation time has improved**
  - now competitive with C backend

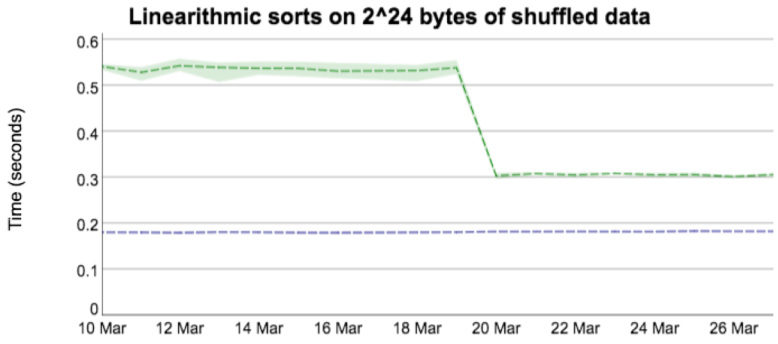


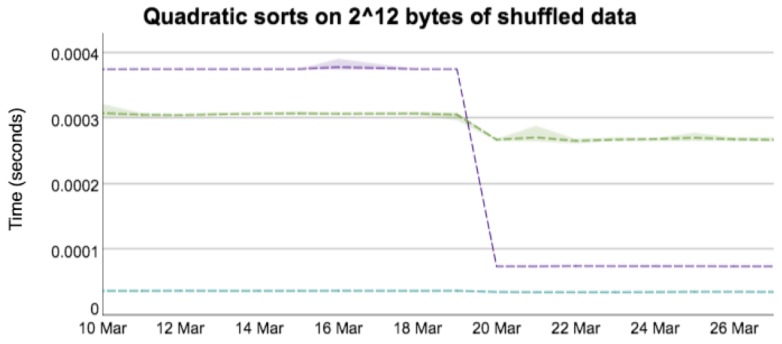
LLVM: Impact: Performance

- --llvm performance has improved with LLVM 6









LLVM Back-end Improvements

Impact:

- LLVM 6.0 vectorizes more cases when llvm-wide-opt is used
- code kept maintainable by removing obsolete version support
- users and developers can start quickly with CHPL_LLVM=system

Status:

- --llvm and --llvm-wide-opt are tested nightly
- performance is improving and generally competitive with C backend
  - occasionally beating C compilers

Next Steps:

- continue to push towards using --llvm by default
- port Chapel's LLVM interface to ARM
  - match ABI characteristics that differ from x86-64

Communication Optimization with --llvm-wide-opt

Comm Opt: Background

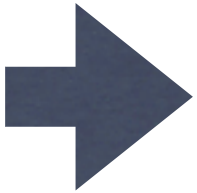
- Idea is to use LLVM passes to optimize GET and PUT
- Enabled with --llvm-wide-opt compiler flag
- First appeared in Chapel 1.8
- Unfortunately was not working in 1.15 and 1.16 releases

Comm Opt: in a Picture

```
// x is possibly remote
var sum = 0;
for i in 1..100 {
  %1 = get(x);
  sum += %1;
}
```

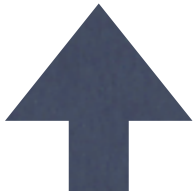
TO GLOBAL MEMORY

```
var sum = 0;
for i in 1..100 {
  %1 = load <100> %x
  sum += %1;
}
```



EXISTING LLVM OPTIMIZATION LICM

```
var sum = 0;
%1 = load <100> %x
for i in 1..100 {
  sum += %1;
}
```



TO DISTRIBUTED MEMORY

```
var sum = 0;
%1 = get(x);
for i in 1..100 {
  sum += %1;
}
```

load <100> %x = load i64 addrspace(100)* %x

Comm Opt: Details

- **Uses existing LLVM passes to optimize GET and PUT**
  - GET/PUT represented as load/store with special pointer type
  - normal LLVM optimizations run and optimize load/store as usual
  - an LLVM pass lowers them back to calls to the Chapel runtime
- **Optimization gains from this strategy can be significant**
  - See "LLVM-based Communication Optimizations for PGAS Programs"
- **Historically, needed packed wide pointers as workaround**
  - wide pointer normally stored as a 128-bit struct: {node id, address}
  - bugs in LLVM prevented using 128-bit pointers
  - packed wide pointers store node id in high bits of a 64-bit address
  - led to scalability constraints — maximum of 65536 nodes
  - sometimes made --llvm-wide-opt code slower than C backend

Comm Opt: This Effort, Impact

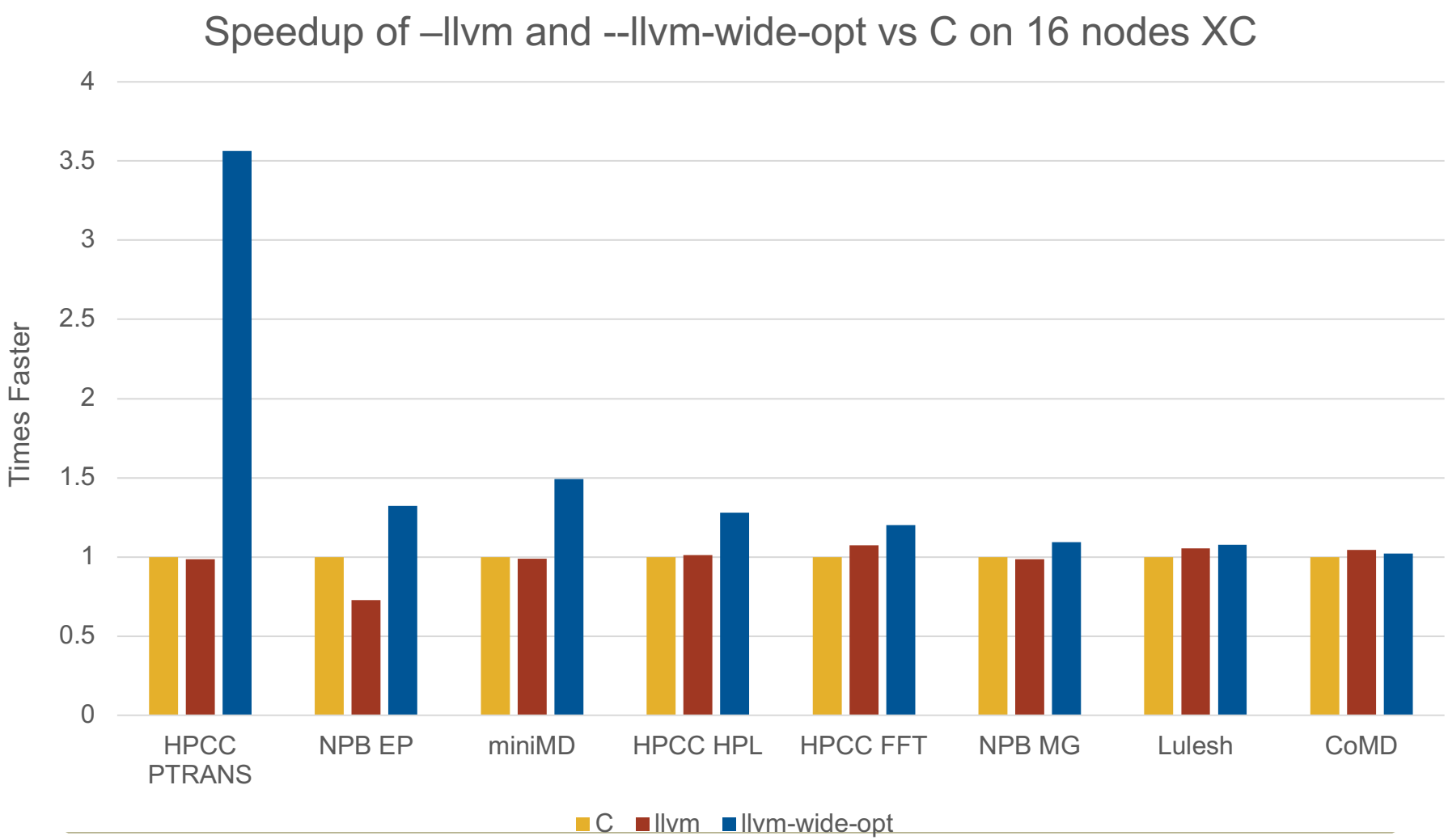
This Effort: Fix --llvm-wide-opt for 1.17

- remove packed wide pointer workaround
- remove CHPL_WIDE_POINTERS configuration variable
- resolve other bugs, including 2 bugs in LLVM itself
- perform initial performance study

Impact: --llvm-wide-opt is much closer to production quality

- Design now supports more than 100,000 nodes
- Overhead is reduced
- No longer reduces performance relative to the C backend
- Significant performance improvement for some benchmarks

Comm Opt: Impact



Comm Opt: Next Steps

- Perform more testing
- Contribute bug fixes for 128-bit pointers upstream
- Enable --llvm-wide-opt by default with --fast
- Reduce compile time spent in this optimization

Other Compiler/Tool Improvements

Other Compiler/Tool Improvements

- Extern blocks now support #defines with casted literals
- Rewrote and improved the `printchplenv` tool
- Rewrote and improved the `compileline` tool
- Added error handling constructs to syntax highlighters
- Added `prototype` modules to syntax highlighters

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY

THE SUPERCOMPUTER COMPANY