Compiler Improvements

Chapel Team, Cray Inc.
Chapel version 1.13
April 7, 2016

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

- Controlling / Exposing Chapel’s Configuration
- Complex Type Improvements via C99 Complex
- Other Compiler Improvements

Controlling / Exposing Chapel's Configuration

Chapel Configuration: Background

Background:

- Chapel's configuration specified by ~23 environment variables
  - CHPL_HOME, CHPL_COMM, CHPL_MEM, CHPL_TASKS, …

- Changing configurations required changing env. vars:

```
export CHPL_COMM=none
chpl foo.chpl -o foo-shared
export CHPL_COMM=gasnet
chpl foo.chpl -o foo-dist
```

- Built-in CHPL_* params have permitted Chapel code to access config.
  - yet never documented with chpldoc

Chapel Configuration: This Effort

- **Added compiler flags for specifying configuration**

```
--home, --comm, --mem, --tasks, …
```

  - Backed by existing CHPL_* environment variables
    - more consistent with other compiler flags
  - Simplifies ability to switch between configurations:

```
chpl --comm=none   foo.chpl -o foo-shared
chpl --comm=gasnet foo.chpl -o foo-dist
```

  - Flags documented in man page and `chpl --help` output

- **Refactored params into their own (internal) module**
  - Location:

```
modules/internal/ChapelEnv.chpl
```

  - Documented with chpldoc, accessible in online documentation: http://chapel.cray.com/docs/1.13/modules/internal/ChapelEnv.html

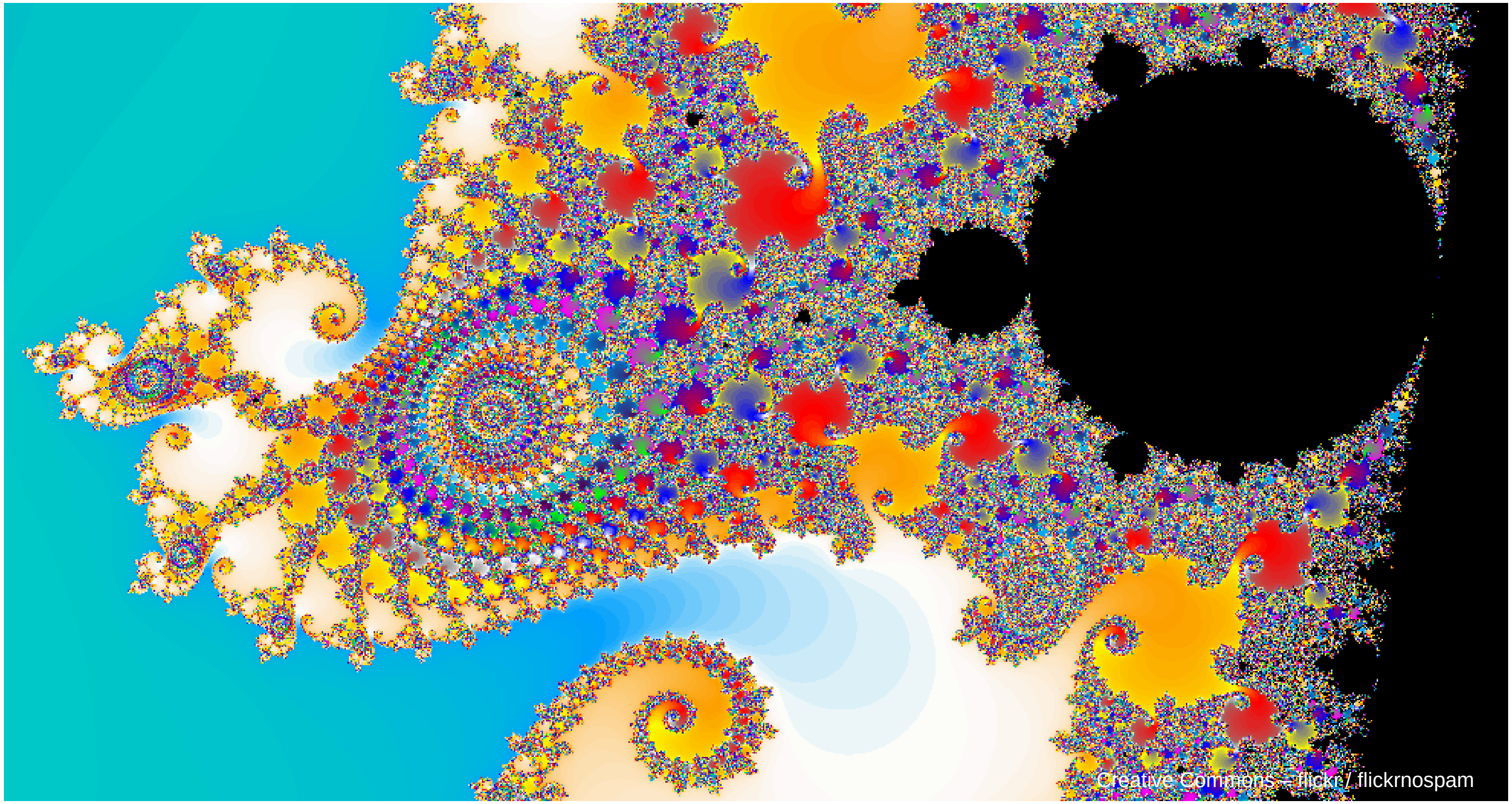
Chapel Configuration: Next Steps

- **.chplrc file**
  - Add ability to specify default Chapel configuration / compilation flags
  - Will benefit package-based deployment and system-wide installations
  - Will simplify managing multiple configurations
- **Rename ChapelEnv.chpl and reword documentation**
  - Name/text reflects historical nature: environment variable-centric

Complex Type Improvements via C99 Complex

Complex Types: Background

- **Complex types were implemented as homegrown records**
  - Stored two *real* valued components
  - Performed math operations on the individual components



Creative Commons – flickr / flickrnospam

Complex Types: This Effort

- **Replace our homegrown records with C99 complex types**
  - Generate cleaner code
  - Generate safer floating point code

- **Give back-end compiler more complete type information**

- **Simplify the compiler implementation**
  - Removed complexToRecord pass

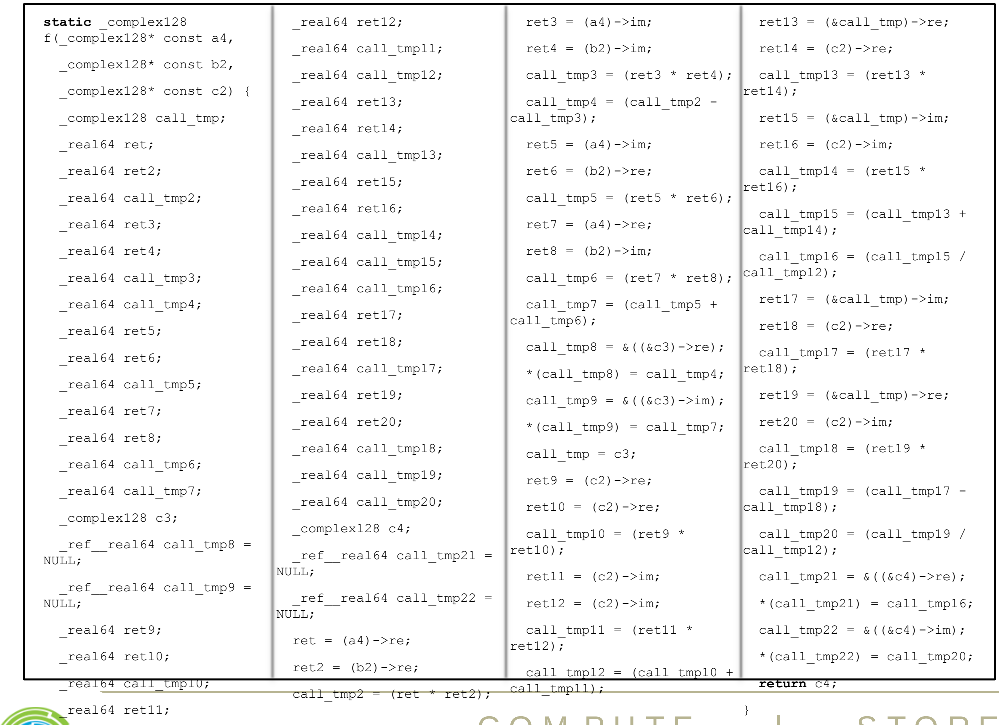
- **Simplify interoperability with C libraries**

Complex Types: Impact

- Improved generated code involving complex types

```
proc f(a: complex, b: complex, c: complex) {
  return a * b / c;
}
```

v1.12

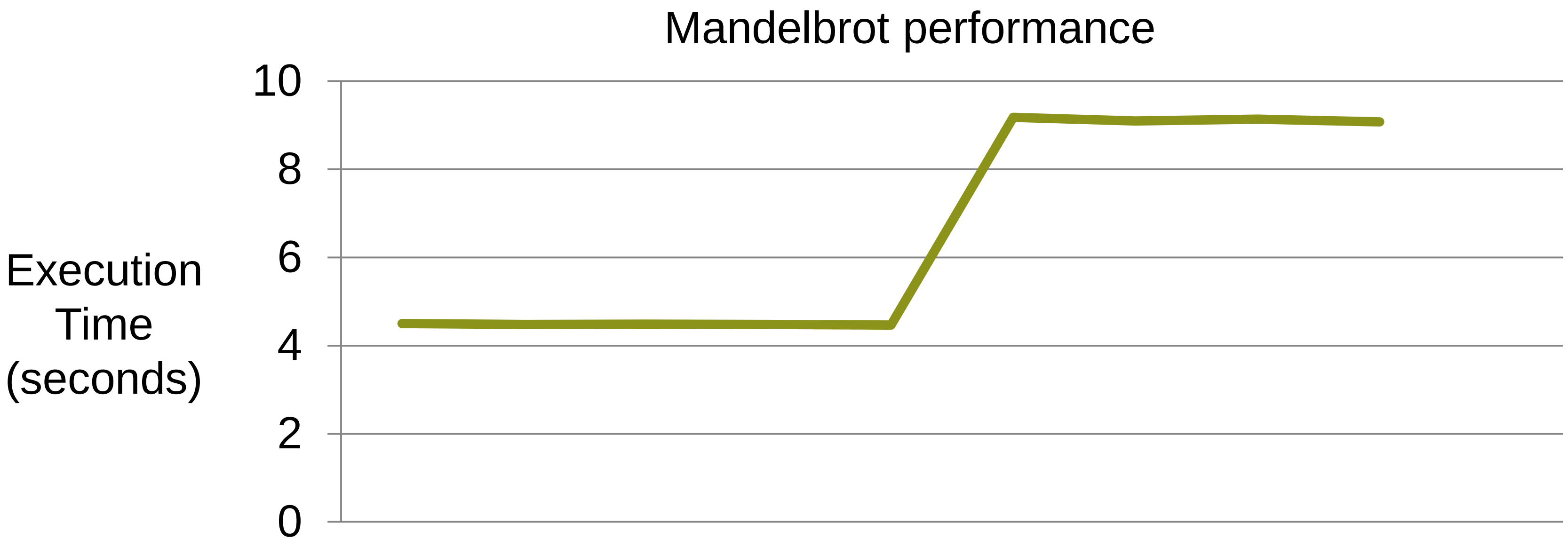


```
static _complex128
f(_complex128* const a4,
  _complex128* const b2,
  _complex128* const c2) {
  _complex128 call_tmp;
  _real64 ret;
  _real64 ret2;
  _real64 call_tmp2;
  _real64 ret3;
  _real64 ret4;
  _real64 call_tmp3;
  _real64 call_tmp4;
  _real64 ret5;
  _real64 ret6;
  _real64 call_tmp5;
  _real64 ret7;
  _real64 ret8;
  _real64 call_tmp6;
  _real64 call_tmp7;
  _complex128 c3;
  _ref__real64 call_tmp8 =
NULL;
  _ref__real64 call_tmp9 =
NULL;
  _real64 ret9;
  _real64 ret10;
  _real64 call_tmp10;
  _real64 ret11;
  _real64 ret12;
  _real64 call_tmp11;
  _real64 call_tmp12;
  _real64 ret13;
  _real64 ret14;
  _real64 call_tmp13;
  _real64 ret15;
  _real64 ret16;
  _real64 call_tmp14;
  _real64 call_tmp15;
  _real64 call_tmp16;
  _real64 ret17;
  _real64 ret18;
  _real64 call_tmp17;
  _real64 ret19;
  _real64 ret20;
  _real64 call_tmp18;
  _real64 call_tmp19;
  _real64 call_tmp20;
  _complex128 c4;
  _ref__real64 call_tmp21 =
NULL;
  _ref__real64 call_tmp22 =
NULL;
  ret = (a4)->re;
  ret2 = (b2)->re;
  call_tmp2 = (ret * ret2);
  ret3 = (a4)->im;
  ret4 = (b2)->im;
  call_tmp3 = (ret3 * ret4);
  call_tmp4 = (call_tmp2 -
call_tmp3);
  ret5 = (a4)->im;
  ret6 = (b2)->re;
  call_tmp5 = (ret5 * ret6);
  ret7 = (a4)->re;
  ret8 = (b2)->im;
  call_tmp6 = (ret7 * ret8);
  call_tmp7 = (call_tmp5 +
call_tmp6);
  call_tmp8 = &((&c3)->re);
  *(call_tmp8) = call_tmp4;
  call_tmp9 = &((&c3)->im);
  *(call_tmp9) = call_tmp7;
  call_tmp = c3;
  ret9 = (c2)->re;
  ret10 = (c2)->re;
  call_tmp10 = (ret9 *
ret10);
  ret11 = (c2)->im;
  ret12 = (c2)->im;
  call_tmp11 = (ret11 *
ret12);
  call_tmp12 = (call_tmp10 +
call_tmp11);
  ret13 = (&call_tmp)->re;
  ret14 = (c2)->re;
  call_tmp13 = (ret13 *
ret14);
  ret15 = (&call_tmp)->im;
  ret16 = (c2)->im;
  call_tmp14 = (ret15 *
ret16);
  call_tmp15 = (call_tmp13 +
call_tmp14);
  call_tmp16 = (call_tmp15 /
call_tmp12);
  ret17 = (&call_tmp)->im;
  ret18 = (c2)->re;
  call_tmp17 = (ret17 *
ret18);
  ret19 = (&call_tmp)->re;
  ret20 = (c2)->im;
  call_tmp18 = (ret19 *
ret20);
  call_tmp19 = (call_tmp17 -
call_tmp18);
  call_tmp20 = (call_tmp19 /
call_tmp12);
  call_tmp21 = &((&c4)->re);
  *(call_tmp21) = call_tmp16;
  call_tmp22 = &((&c4)->im);
  *(call_tmp22) = call_tmp20;
  return c4;
}
```

v1.13

```
static _complex128 f(_complex128 a4,
                     _complex128 b2,
                     _complex128 c2) {
  _complex128 call_tmp;
  _complex128 call_tmp2;
  call_tmp = (a4 * b2);
  call_tmp2 = (call_tmp / c2);
  return call_tmp2;
}
```

Complex Types: Performance-Safety Tradeoff



- **Caused by back-end multiply and divide implementation**
  - Safer for “weird” floating point values, but slower overall
    - NaNs
    - Infinities
    - Denormalized
- **Compile with ‘--no-ieee-floats’ to regain the performance**

Other Compiler Improvements

Other Compiler Improvements

- **enum config values can now be fully qualified**
  (contributed by Nick Park)
  - supports,

    ```
    ./a.out --fillColor=color.red
    ```

  - whereas previously, one had to use:

    ```
    ./a.out --fillColor=red
    ```

- **the --ldflags flag now “stacks” when several are used**
- **the --no-warnings flag can now be reversed via --warnings**
- **improved filename representation in generated code**
  - switched from string literals to a table of filenames + indices into it
- **also:**
  - many performance improvements (covered elsewhere)
  - many developer-oriented improvements (see CHANGES.md)

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 CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY