Compiler / Implementation

Chapel Team, Cray Inc.

Chapel version 1.15

April 6, 2017

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

- Const-checking Improvements
- New Execution Time Checks
- Denormalize Pass On By Default
- Printing the Static Call Graph
- Error Message Improvements
- Bug Fixes

Const-checking Improvements

Const-checking: Background

- **Chapel's const-checking has been incomplete**
- **Problematic areas:**
  - blank-intent array arguments
  - methods that set 'this' or a field
- **These examples should report an error but compile in 1.14**

```
proc setOne(A) {
  A[1] = 1;
}
const A: [1..4] int;
setOne(A); // 1.14: missing error
```

```
record R {
  var x: int;
  proc reset() {
    this.x = 0;
  }
}
const r = new R(1);
r.reset(); // 1.14: missing error
```

Const-checking: This Effort

- **Adds new const-checking implementation**
- **Better handling of many difficult cases**
  - return intent overloads
  - array blank intent and record 'this' blank intent
  - field access
- **The examples now report a compilation error**

```
proc setOne(A) {
  A[1] = 1;
}
const A: [1..4] int;
setOne(A); // 1.15: compilation error
```

```
record R {
  var x: int;
  proc reset() {
    this.x = 0;
  }
}
const r = new R(1);
r.reset(); // 1.15: compilation error
```

Const-checking: Impact and Next Steps

Impact:

- New checking revealed a significant number of errors
  - in both standard modules and in tests
- Fixing these improved the quality of module and test code

Next Steps:

- Address problem areas in the improved implementation
  - if expressions
  - tuples, especially nested tuples

New Execution Time Checks

Exec-time checks: Divide by Zero

Background:

- Chapel supports many execution-time checks to avoid exceptions
  - goal: users should not typically see segfaults, floating point exceptions, etc.
- checks are on by default to support productivity
  - can be disabled for performance runs (e.g., --no-bounds-checks, --fast)
- Recently, a few users have run into faults due to integer divide-by-zero

This Effort:

- add execution time checks to guard against integral divide-by-zero
  - floating point types behave as always
- add new flag to enable/disable these checks:

  `--[no-]div-by-zero-checks`
  - also controlled by '--[no-]checks' and '–fast'

Impact: users now get better error messages in such cases

Exec-time checks: empty ‘Block’ bounding box

Background:

- Chapel’s ‘Block’ distribution takes a domain bounding box argument
  - indicates which indices define the block distribution
- If domain was empty / uninitialized, caused a divide-by-zero error
- Check on previous slide now catches divide-by-zero
  - but root cause would be unclear to a typical user (“I wasn’t dividing…”)

This Effort:

- add execution time checks to guard against such cases:

```
error: halt reached - Block() requires a non-empty boundingBox
```

- always enabled: ‘Block’ distributions are already expensive to set up
  - overhead of new check unlikely to be noticeable

Impact: users will now get better error messages for such cases

Exec-time checks: Next Steps

- Continue to improve messages for confusing error cases
- As error-handling feature matures, use it for such cases

Denormalize Pass On By Default

Denormalize: Background

- **Introduced 'denormalize' compiler pass in 1.14**
  - Goal: make generated C code easier to read
  - Disabled by default due to insufficient time for testing prior to release
    - Controlled with "--[no-]denormalize"

- **Consider this Chapel code:**

```
config var x = 123;
writeln(x*x + x);
```

- **Generated C without denormalize:**

```
int tempA = x * x;
int tempB = tempA + x;
writeln_chpl(tempB);
```

- **With denormalize:**

```
writeln_chpl(x*x + x);
```

Denormalize: This Effort and Next Steps

This Effort:

- Enabled denormalize pass by default
- Fixed minor bugs revealed by further testing

Impact:

- Cleaner code for developers to read

Next Steps:

- Clean up more cases in generated code
  - e.g. unnecessary casts, parentheses

Printing the Static Call Graph

Printing the Static Call Graph

Background: It can be useful to see the call graph of a program

- to understand which functions call other functions
  - including both direct calls and indirect calls through other functions

This Effort: Add --print-callgraph compiler option

- prints full static call graph starting from program entry point

Impact: Helps to understand the flow of a program at a glance

--print-callgraph output for ptrans.chpl example benchmark

```
main
  printConfiguration
    printProblemSize
  initArrays
  verifyResults
    CPlusATranspose
    chpl__reduce7_eltype
  printResults
```

Error Message Improvements

Error Message Improvements

- Re-assigning a ‘param’ after it is initialized
- Indicate illegal ‘param’ types more clearly
- Print multiple ‘param’ errors before halting compilation
- Applying ‘inline’ to a recursive function
- Trying to cast to a value rather than a type
- Trying to capture a generic function
- An initializer that attempts to return a value
- Provide a clear message if a copy initializer is needed but is not defined
- Using domain queries in field declarations
- Querying the ‘IRV’ of a non-sparse array
- Applying vector ops to non-1D rectangular arrays
- Removed warning for some cases of serialized assignments
- Improvements to messages for $CHPL_HOME/util/chplenv/ scripts

Bug Fixes

Bug Fixes

- Multiple bugs relating to array-of-array and sparse-array semantics
- Reading null bytes into strings when lengths are specified
- Resetting size when clearing sparse block-distributed domains
- Failure to resolve ‘uint **= uint’
- Support for ‘extern’ blocks
- Wrong LD was selected by our Makefiles
- Buffer overflow problem in the parser for long function signatures
- Broken link to Quickstart.rst
- LLVM back-end couldn’t support multiple --ccflags
- Off-by-one bug in string.split()
- Extern procedures returning ‘void’
- Arrays of c_strings in --no-local compilation mode

More Bug Fixes

- Order of module-scope variable deinitializations
- Generalized the find() routine on arrays to non-1D arrays
- Type methods did not support default arguments well
- Pop_front() could cause the array to grow
- The --cpp-lines flag was sometimes ignored
- fixed a bug in ‘Spawn’ when one sub-process consumes another's output
- An error in BlockCyclic indexing
- User-defined initializers wouldn’t accept array fields
- Error creating virtual dispatch tables for generic class inheritance
- Recursive iterator inlining
- An error supporting ‘stridable’ queries on Replicated arrays
- A copy propagation bug related to array allocation

Even More Bug Fixes

- A ‘chplvis’ bug that caused segementation faults
- Type methods could be overly generic w.r.t. their receiver
- An error casting reference expressions
- Fixed a race condition in initializing locale models
- Fixed bugs in the standalone parallel iterator for CSR domains
- Fixed a bug related to module-scoped ‘ref’ declarations
- Fixed a bug in which the number of CPUs was sometimes reported to be 0
- Fixed bugs in the ‘localSubdomain*’ calls for local arrays/domains

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