Language Improvements

Chapel Team, Cray Inc. Chapel version 1.17 May 9, 2018



COMPUTE | STORE | ANALYZE

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

• Initializers

- Improvements to the Proposal
- <u>Compiler-Generated Initializers</u>
- Other Changes of Note
- Overall Status and Next Steps
- Error Handling
- Argument Intent Changes
- Improving Productivity of 'delete'
- Accessing Type and Param Fields
- <u>Numeric Coercion Improvements</u>
- Early Exits from 'forall'
- <u>"""Uninterpreted String Literals""</u>
- Other Language Improvements



COMPUTE | STORE |

ΑΝΑLΥΖΕ



N

Initializers



COMPUTE | STORE | ANALYZE



Initializers: Background and Summary of Work

Background:

- Have been developing initializers to replace constructors
 - Provide significantly more control over classes and records
 - Extensive progress made over last few releases
- As of Chapel 1.16...
 - Not all features implemented
 - Some open questions remained
 - Some behavior was not ideal

This Effort:

- Revisited the proposal, based on experience using initializers
- Improved support for compiler-generated initializers



Initializers: Outline

- Improvements to the Proposal
- Compiler-Generated Initializers
- Other Changes of Note
 - Copy Initializers
 - Operations on Initialized Fields
 - Select Bug Fixes

Overall Status and Next Steps



COMPUTE | STORE | ANALYZE



Improvements to the Proposal



COMPUTE | STORE | ANALYZE

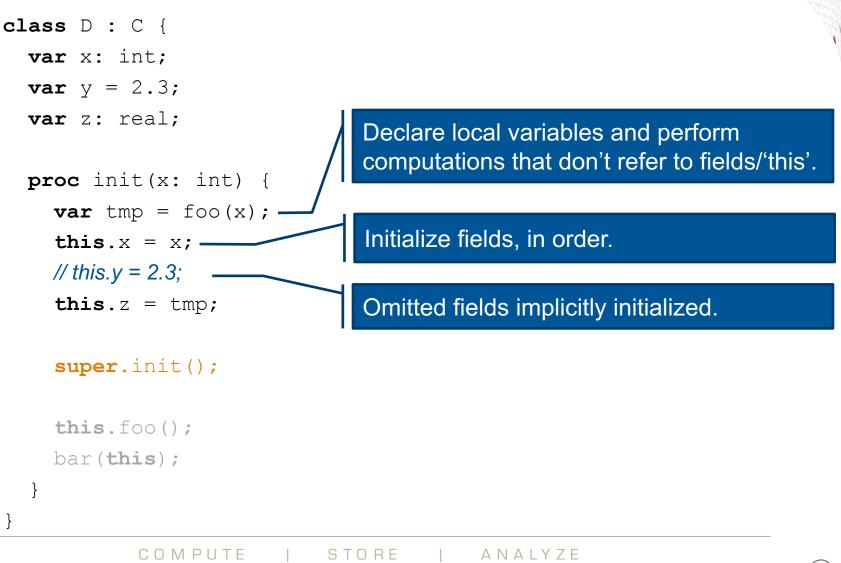


Initializers: Old Proposal

- Given a class hierarchy:
 - Classes A through D form a hierarchy: D:C:B:A
- Each class implements one or more 'init()' methods
- Body of 'init()' was divided into two phases
- In phase 1, object was uninitialized memory
 - Couldn't do much with it other than initialize its fields
- In phase 2, object was a D (for any initializer)
 - Could do anything with it



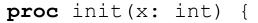
Old Proposal: Phase 1, things you could do





Old Proposal: Phase 1, things you couldn't do

```
class D : C {
    var x: int;
    var y = 2.3;
    var z: real;
```



var tmp = foo(x);
this.x = x;
// this.y = 2.3;
this.z = tmp;

COMPUTE

Couldn't call methods or refer to parent fields. *Rationale:* parent fields are not initialized yet.

```
super.init(...) invokes the parent initializer.
```

bar(this);

this.foo();

super.init();

STORE | A

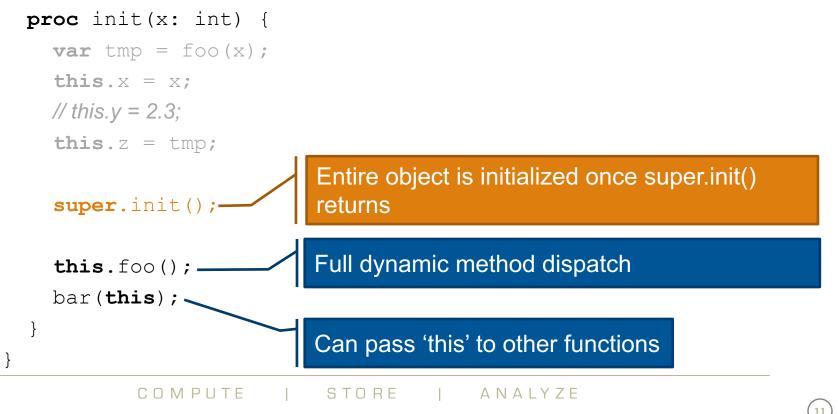






Old Proposal: Phase 2, things you could do

```
class D : C {
    var x: int;
    var y = 2.3;
    var z: real;
```



Old Proposal: The Big Problem

```
class AbstractArr {
  param rank: int;
  proc init(param rank: int) {
    this.rank = rank;
    super.init();
  }
}
```

class RectangularArr : AbstractArr {
 var bounds: rank*int;

proc init(bounds...) {
 // problem: can't set or use 'bounds' field
 // because 'rank' is not yet established
 this.bounds = bounds;
 super.init(bounds.size)



STORE

}

ANALYZE

Old Proposal: Other design Qs to revisit

• Phase 1 or phase 2 by default if no 'super.init()'?

- Originally chose phase 2 as default
- Needed to call 'super.init()' to initialize const/param/type fields

• 'super.init()' as phase 1 vs. 2 separator

- Records don't inherit, so don't have a 'super'
 - Yet still required its use in order to specify phase 1 actions

Modest interest in old-style 'initialize()' methods

- A hook called after constructor
- Convenient way to leverage compiler-generated constructor

New proposal also helps with each of these issues.



New Proposal: Overview

Parent fields initialized before child fields

Can now use parent fields to initialize child fields
 var bounds : rank*int; // OK!

• Can call methods on parent type in 'init()'

- Current type's methods can be called...
 - ... after a 'this.complete()' call or
 - ... after a 'this.init()' call

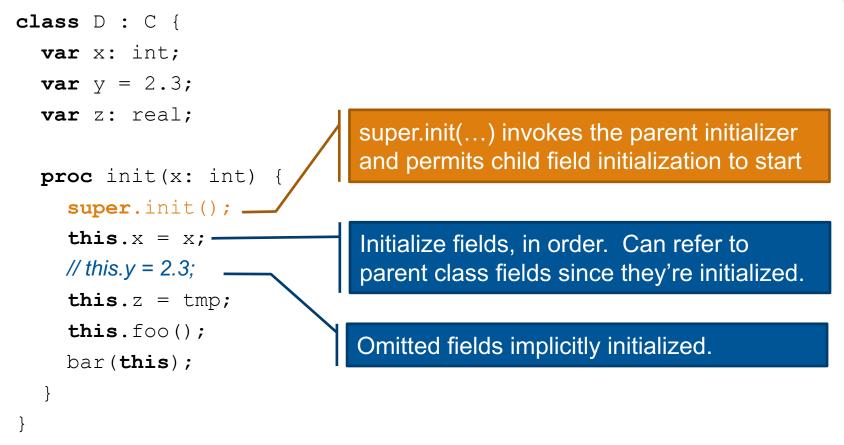
Introduces 'postinit()' as replacement for 'initialize()'





New Proposal: init() overview

• 'super.init()' called at start rather than end of phase 1

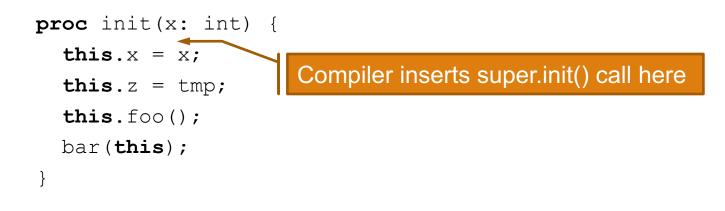




New Proposal: init() details

• Details:

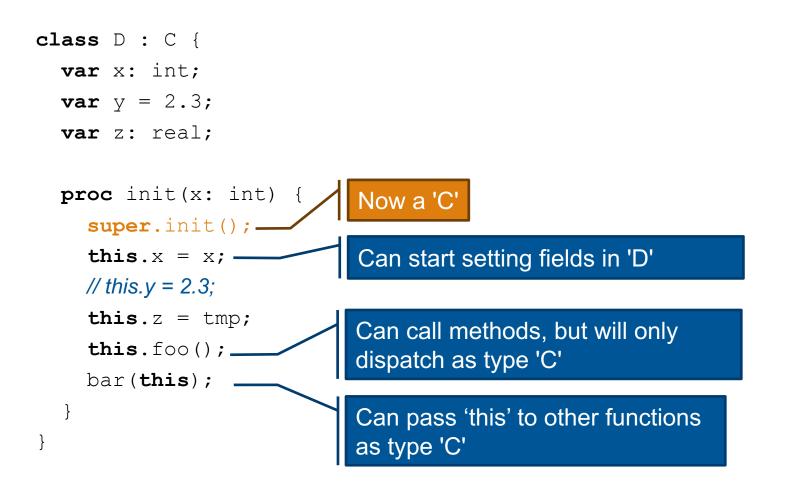
• If super.init(...) is omitted, compiler inserts 0-arg super.init() call at top



- Records no longer support super.init()
 - They don't need to since it's not used as a separator anymore
 - Consistent with not supporting record inheritance



New Proposal: init() overview





New Proposal: this.complete() Support a way to initialize remaining fields class D : C { proc init(x: int) { Transitions object from a 'C' to a 'D' this.x = x; this.complete(); Subsequent method calls could dispatch to a this.foo(); method defined on 'D' or its parents bar(this);

Enables method calls within record init()s



}

COMPUTE | STORE | ANALYZE

Copyright 2018 Cray Inc.

it is a 'D' object

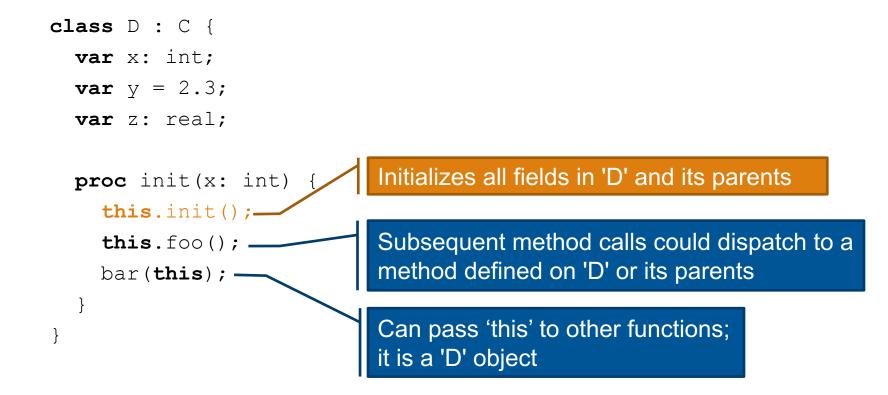
Can pass 'this' to other functions;



New Proposal: this.init()

• this.init(): Similar to use in the 1.16 release

• Calls another 'init()' defined on D





COMPUTE | STORE | ANALYZE

New Proposal: Summary of init()

- Given a class hierarchy:
 - Classes A through D form a hierarchy: D:C:B:A

In D.init(), object starts as nothing (a blob of memory)

• Implication: You can't do much with it yet

• After D's call to super.init(), object is a C

- Implication: You can do anything with it that you could do with a C
- Plus, you can also assign to D fields to help turn it into a D

• Object becomes a D:

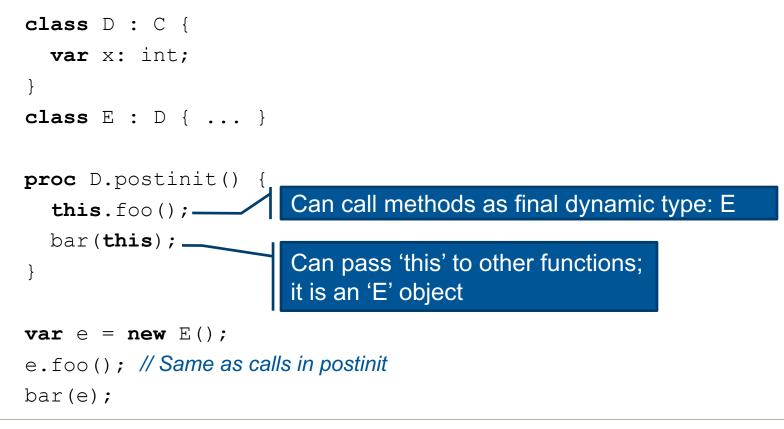
- After D's call to D.complete(), or
- After D's call to this.init(), or
- After D.init() returns



New Proposal: postinit() overview

• postinit(): A hook called after initialization

- Convenient way to leverage default initializers
- Supports virtual dispatch into child methods at object creation time





ANALYZE

New Proposal: postinit() details

• Details:

- postinit() takes no arguments
- If postinit() is not defined for a class, compiler inserts:
 proc postinit() {

```
super.postinit();
```

• Compiler inserts super.postinit() if omitted in user-written postinit()





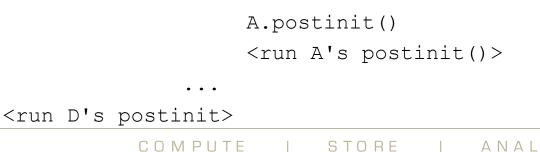
New Proposal: Summary

Parent fields initialized before child fields

```
D.init()
      C.init()
            B.init()
                  A.init()
                   <Initialize A fields>
            <Initialize B fields>
      <Initialize C fields>
<Initialize D fields>
```

. . .

Optional 'postinit()' method called after all init() methods D.postinit()





ANALYZE

Compiler-Generated Initializers



COMPUTE | STORE | ANALYZE



Compiler-Generated Initializers: Background

Last release added support for compiler-generated 'init()'

- Behavior similar to compiler-generated constructors
- Off by default, enabled via developer flag
 - Only applied to classes in user-defined modules
 - Never applied to types with explicit initializers



Compiler-Generated Initializers: This Effort

- Initial support for records in user-defined modules
- Added pragma to apply to individual types
 - To support converting module types with inheritance
 - Will not be needed once enabled by default

Improved error checking for intermixed hierarchies

Inheritance hierarchies with constructors cannot generate 'init'

```
class A {
   proc A (...) {
      // explicit constructor
   }
```

```
pragma "use default init"
class B : A {...}
// error: asks for compiler-generated initializer
```

// but inherits from type with explicit constructor



Compiler-Generated Initializers: Status

• Many bugs fixed, others remain:

- Some expressions cannot be used as default values for fields yet
 - E.g., parallel loops, conditional expressions
- Nested types, when either type is generic, cannot be used
- Fields that are arrays of syncs can cause deadlocks
- Internal compiler errors

Once these bugs are resolved, can generate by default

• And deprecate constructors







Initializers: Other Changes of Note



COMPUTE | STORE | ANALYZE



Initializers: Copy Initializers

Generic 1-arg init() now recognized as potential copy init()

- Compiler warns user of this subtlety for related compilation errors
- Can avoid warnings with explicit type or a where clause record Foo {

```
proc init(x: Foo) { ... } // Actual copy init
```

• May evolve this design further to make copy initializers clearer

Compiler now generates copy initializer if no match found

• Open Question: When user defines copy init or assignment, should compiler attempt to define the other based on it? Should it warn?



}





Initializers: This Effort

Support more operations on initialized fields

• Can reassign field once initialized

```
this.x = 5;
this.x *= 2; // Now allowed in 1.17
```

- Can pass a field as an argument to a function this.y = "hello"; writeln(this.y);
- Still an error to initialize fields out of order this.secondField = 5; this.firstField = 10; // Error!



Initializers: This Effort

• Other bug fixes

- Enabled support for promotion over types with initializers
- 'new D(...)' only calls 'D.init(...)'
 - Won't dispatch to parent class initializer with similar argument list
 - Avoids hiding compiler-generated initializer when parent has explicit 'init()'
- Allow fields to infer their type when default value is a 'new' expression var myField = new D();
- Many others (see <u>CHANGES.md</u> file for details)







Initializers: Overall Status and Next Steps



COMPUTE | STORE | ANALYZE



Initializers: Status

Most library/internal modules converted to initializers

- Exceptions:
 - Arrays, domains, distributions: issue with using inherited field, now resolved
 - Owned, Shared, strings: special initCopy/autoCopy functions
 - Reductions: compiler still generates constructors by default

Most tests converted to initializers

- Out of ~8,500 tests...
 - ... 26 remain unconverted due to bugs or unimplemented features
 - ... 28 others will be removed once constructors are deprecated



Initializers: Next Steps

Finish compiler-generated initializers

• Fix bugs

- Nested types when at least one of the types is generic
- Generic instantiation when generic fields initialized in conditional

• ...

Deprecate constructors

• Finalize design decisions:

- Finalize copy initializers
- Finalize type initializer story
- Allow users to opt into retaining compiler's default 'init()'?
 - Currently squashed by user's 'init()'

Support incomplete initialization when explicitly requested

Also known as the 'noinit' feature





N

Error Handling



COMPUTE | STORE | ANALYZE



Error Handling: Background

Error handling helps users with exceptional cases

• For example, handling a failure when opening a file:

```
var f: file;
```

```
try {
```

}

```
f = open(f1, iomode.r); // if open() raises an error, jump to the catch block
writeln("everything is fine");
```

```
} catch {
```

```
writeln("an error occurred"); // catch blocks are used to handle errors
```



Error Handling: Background

Greatly improved in previous releases

- Supported in parallel and multi-node code
- Fine-grained error checking modes
- 'SystemError' hierarchy provided for common error cases

• But as of 1.16, standard modules still halt in many cases

• Highly problematic for users and library writers



Error Handling: This Effort

- Exclude throwing from 'defer', 'deinit()'
- Use error handling more in the standard library
- Bug fixes



COMPUTE | STORE | ANALYZE

Exclude throwing from 'defer', 'deinit()'

Initially considered throwing from 'defer', 'deinit()'

var f: file = open(...);
defer try f.close();

 But that could prevent other 'defer', 'deinit()' from running defer thisNeedsToHappen(); // will this run if f.close() throws?
 defer try f.close(); // what is the handling context of this block?

• Also, no clear way to handle such an error



Exclude throwing from 'defer', 'deinit()'

'defer', 'deinit()' must now handle errors internally
 defer {

```
try {
  f.close();
} catch e { // suggested pattern: complete handling, logging
  logError(e.message());
}
```



}

Use error handling in internal and library code

• Before, illegal cast operations would halt:

```
var s = "brad";
```

var i = s: int;

> error: Unexpected character when converting from string to int(64): 'b'



COMPUTE | STORE | ANALYZE



Use error handling in internal and library code

```
• Now it throws an 'IllegalArgumentError':
```

```
var s = "brad";
try! {
  var i = s: int;
} catch e: IllegalArgumentError {
  writeln("caught cast error");
}
> caught cast error
```

• Addressed several other halts in standard library modules





Correctly enforced error handling rules in 'coforall' loops

```
proc test() {
  coforall i in 1..10 {
    throwme(); // throwme() is unhandled and 'test()' does not throw
    try! { } // but this empty try! made it pass error checking
    } // now a compilation error as intended
}
```

Fixed garbage memory returns from 'try'/'catch'



Error Handling: Status and Next Steps

Status:

• Error handling is increasingly ready for production code

Next Steps:

- Implement missing features
 - Throwing from 'init()'
 - Throwing from non-inlined iterators
- Use error handling where appropriate in library modules
 - Deprecate 'out error' pattern
 - Wherever reasonable, remove 'halt()'
- Explore lower-overhead implementations of error-handling
 - E.g., avoid conditionals for non-error cases







COMPUTE | STORE | ANALYZE



Argument Intent Changes

This Effort: Improved several kinds of argument intents

- <u>'in' intent for functions</u>
- in' intent for tasks
- range default intent
- <u>'type' intent</u>

Impact: Intents are more flexible and consistent

Status: Improvements implemented and specified

Next steps:

- adjust default initializers to use 'in' intent and avoid copies
- improve 'out' and 'inout' intents



'in' Intent: Background

'in' intent always created a copy

contrast with variable initialization

var x = g(); // does not create a copy if g returns record by value



Copyright 2018 Cray Inc.



COMPUTE | STORE

ANALYZE

'in' Intent: This Effort

Make 'in' intent more similar to variable initialization

```
// before 1.17
                                       // after 1.17
record R { var x: int }
                                       record R { var x: int }
var globalR: R;
                                       var globalR: R;
proc f(in x) { }
                                       proc f(in x) { }
f(new R(1));
                                       f(new R(1));
f(globalR);
                                       f(globalR);
                                       proc f(ref x) {
proc f(in x) {
  var x tmp = copy-init x;
                                         deinit x;
  deinit x tmp;
                                       }
}
var call tmp = new R(1);
                                       var call tmp = new R(1);
f(call tmp);
                                       f(call tmp);
deinit call tmp;
                                       var x tmp = copy-init globalR;
f(qlobalR);
                                       f(x tmp);
```



COMPUTE

ANALYZE

Copyright 2018 Cray Inc.

STORE

'in' Intent: Impact

- 'in' intent better optimized
- addresses an issue with 'Owned'

```
// before 1.17
record R { var x: int }
var globalR: R;
proc f(in x) { }
f(new R(1));
f(globalR);
proc f(in x) {
  var x tmp = copy-init x;
  deinit x tmp;
var call tmp = new R(1);
f(call tmp);
deinit call tmp;
f(qlobalR);
```

Copy no longer generated for 'f(new R(1))'

```
// after 1.17
record R { var x: int }
var globalR: R;
proc f(in x) { }
f(new R(1));
f(globalR);
proc f(ref x) {
  deinit x;
}
var call tmp = new R(1);
f(call tmp);
var x tmp = copy-init globalR;
f(x tmp);
```



ANALYZE

'in' Intent for Tasks

Background: 'in' task intent was handled after task launch

- Causing the potential for race conditions when combining:
 - record copy initializer
 - 'begin'
 - 'in' task intent

This Effort: Handle 'in' task intent during task setup

 Resolves the potential for race condition in a case like the following: record R { /* includes class fields */ }

```
R.init(from: R) { /* copy initialize copies class fields */ }
var r: R;
begin with (in r) {
    f(r);
}
mutate(r); // mutation races with copy from 'in' intent
```

Impact: Potential race condition addressed



Range Default Intent

Background: Before 1.17, range default intent was inconsistent

- for tasks, it was 'const in'
- for functions, it was 'const ref'

This Effort: Changed range default intent to 'const in'

range now behaves more like 'int'

Impact: Range semantics simplified and more optimizable



COMPUTE | STORE | ANALYZE



'type' intent

Background: Combining 'type' intent with type specifier allowed

• e.g.

```
proc f(type t: integral) { }
```

• but behavior of such code was neither specified nor consistent

This Effort: Specify the behavior and address bugs

- 'type' intents with type specifier:
 - limits the 'type' arguments that can be passed in
 - does not allow coercion

Impact: 'type' intent with specified type now usable



Improving Productivity of 'delete'



COMPUTE | STORE | ANALYZE



Productive 'delete': Background and This Effort

Background:

- Previously, 'delete' could only be applied to a single class object
- This made certain patterns verbose:
 - deleting multiple objects:
 - delete C1;
 - delete C2;
 - delete C3;
 - deleting arrays of objects:
 - forall c in Arr do
 - delete c;

This Effort: Improved 'delete' to support...

- ...comma-separated expressions
- ...arrays



Productive 'delete': Impact and Next Steps

Impact:

- Can now write these patterns more succinctly:
 - deleting multiple objects:

delete C1; delete C1, C2, C3; delete C2; delete C3; • deleting arrays of objects: forall c in Arr do delete Arr; delete c;

Next Steps:

- Add support for users to define types that can promote, like arrays
 - ensure that this feature works for such cases



Accessing Type and Param Fields



COMPUTE | STORE | ANALYZE



Accessing Type and Param Fields

Background: classes/records can have 'type' or 'param' fields

- Such fields make the class/record generic
- They are part of the generic type's instantiation
- But, they could not be accessed from a type variable

This Effort: Enable accessing such fields from a type variable

```
record Element { param p; type t; }
type MyElement = Element(1, int);
param MyElementP = MyElement.p;
type MyElementT = MyElement.t;
writeln(MyElementP, " ", MyElementT:string);
// now outputs: 1 int(64)
```

Impact: Type variables and arguments are more capable

• Can now call '.size' on a tuple type



Numeric Coercion Improvements



COMPUTE | STORE | ANALYZE



Numeric Coercions

Background: There are coercions between some numeric types

- But the implementation presented usability issues with 'real(32)'
- For example, each of these lines caused a compilation error:

```
var a: real(32) = 0.0;
var b: real(32) = 1;
var half: real(32) = 1 / (2.0:real(32));
```

This Effort: Improved numeric coercions

 Above 'real(32)' examples now compile and run, as does this example: param x: int(16) = 0; var y: int(8) = x;

Impact: Numeric types of non-default sizes are easier to use







COMPUTE | STORE | ANALYZE



Early Exits from 'forall'

Background: early exit errors were incomplete and unclear **This Effort:** completed checking with clear error messages

```
label outer for ... {
    forall ... {
        continue;
                               // OK: skips the current iteration of 'forall'
        break;
                               // Error: cannot exit a 'forall' from its loop body
                               // Error: cannot exit a 'forall' from its loop body
        return ...;
        continue outer; // Error: cannot exit a 'forall' from its loop body
                               // OK only within the definition of a parallel iterator
        yield ...;
        for ... {
                               // OK: exits the inner for-loop, stays in the forall-loop
            break;
                               // Still an Error: cannot exit a 'forall' from its loop body
             return;
```



"""Uninterpreted String Literals"""



COMPUTE | STORE | ANALYZE



Uninterpreted Strings: Background, This Effort

Background: String literals are interpreted

- e.g. "\n" translates into a newline
- Applies to both 'single' and "double" quote variants
- Literal newlines are not allowed

This Effort: Add triple-quoted uninterpreted string literals

- Uninterpreted e.g. """\n""" is two characters
- Applies to both "single" and ""double" quote variants
- Literal newlines are allowed inside them



Uninterpreted Strings: Impact, Next Steps

Impact: Uninterpreted multi-line strings are available

capturing code as a string

```
var query = """
SELECT
   a_column
, another_column
FROM
   {%s}
WHERE
   {%s} = {%s};""";
```

general purpose multiline messages

```
var helpMsg = """
Usage: ./parallelProg <options>
--option1 : Do option1 things
--option2 : Do option2 things
--option3 : Do option3 things""";
```

Next Steps:

- Consider support for multi-line traditional strings
- Add library functions to trim leading whitespace







COMPUTE | STORE | ANALYZE



Other Language Improvements

Arrays

- 'clear()' on an array of records now calls the records' deinitializers
- Improved support for casting arrays to strings

Domains

- Associative domains may use index types containing ranges
- 'isEmpty()' method on domains
- Subtype queries on distributions now supported

Alignment of non-stridable range is low bound

Used to always be 0





Other Language Improvements

Forwarding

- Error-handling propagates through forwarded methods
- Support for forwarding methods on arrays, domains, and distributions
- See documentation

• Owned and Shared

- <u>'Owned(C)' and 'Shared(C)' coerce to type 'C'</u>
- <u>'Owned(D)' coerces to 'Owned(C)' when D is a subclass of C</u>
- Writing out an 'Owned(C)' simply prints the 'C' object



Other Language Improvements

Miscellaneous

- Recursive parallel iterators may be invoked via 'forall' loop
- Improved support for enums with non-trivial init expressions
- Improved default argument handling
- Support for defining multiple config types in a single statement
- Enabled wide pointers to be cast to 'c_void_ptr'



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.





