Library Improvements

Chapel version 1.20
September 19, 2019

chapel_info@cray.com
chapel-lang.org
@ChapelLanguage
Outline

• New Modules
  • Collection Types
  • UnitTest
  • EpochManager

• Module Improvements
  • Sort
  • Reflection
  • LinearAlgebra
New Modules

• Collection Types
• UnitTest
• EpochManager
Collection Types

List, Map and Set
Collections: Background

• List, set, and map operations had been supported by arrays and domains

```
// Append to a "list"
var D1 = {1..0};
var A: [D1] real;
A.push_back(1.1);

// Add to a "map"
var D2: domain(string);
var B: [D2] int;
B["hello"] = 42;
```
Collections: Background

• Using arrays as collection types had some problems
  • Required (via assertion) that the array and its domain had a 1:1 relationship
  • resulted in surprising instabilities for users once domains were shared

```javascript
// Append to a "list"
var D1 = {1..0};
var A: [D1] real;
A.push_back(1.1); // OK
var C = A;
A.push_back(2.2); // error

// Add to a "map"
var D2: domain(string);
var B: [D2] int;
B["hello"] = 42; // OK
var D = B;
B["bye"] = 33; // error
```
Collections: Background

• Using arrays as collection types had some problems
  • Permitted modifying ‘const’ domains

```plaintext
// Illegally modifies D1
const D1 = {1..0};
var A: [D1] real;
A.push_back(1.1);

// Illegally modifies D2
const D2: domain(string);
var B: [D2] int;
B["hello"] = 42;
```
Collections: This Effort

- Implemented new collection types
  - list – Replaces "vector-like" operations on arrays
  - map – Replaces "map-like" operations on arrays
  - set – Alternative to set operations on associative domains

- Deprecated list- and map-style operations on arrays
  - pop_back, push_back, pop_front, push_front
  - clear, insert, remove
  - |=, +=, ^=, &=
The list type enables users to build up and iterate over a collection of elements

```c
use List;

var lst: list(int);    // Declare a list
lst.extend(1..8);     // Extend with items from a range.
writeln(lst);
```

Parallel-safe operations can be used by setting "parSafe" to true at initialization

```c
var lst: list(int, parSafe=true);
coforall tid in 1..8 with (ref lst) do
    lst.append(tid);
```
Collections: List

- Intended to replace "vector-like" array methods, which are now deprecated

  ```
  var foo: [1..0] int;
  foo.push_back(4); // warning: push_back is deprecated – please use list.append
  foo.clear(); // warning: clear is deprecated – please use list.clear
  ```

- Can now be written as:

  ```
  use List;
  var foo: list(int);
  foo.append(100);
  foo.clear();
  ```
Collections: Map

• The map type enables users to associate keys with values. It supports…
  • Assignment to any index of the key type
  • Reading from indices in the map
  • Iterators over the keys, values, or key-value pairs
  • Boolean set operators over the keys

```
// Declarations and basic usage
use Map;
var m1 = new map(string, int);
var m2 = new map(string, int);
m1["one"] = 1;
m2["two"] = 2;

// Iterators
for k in m1 {...} // keys
for v in m1.values() {...} // values
for kv in m1.items() {...} // both
```

```
// Boolean set operators
m1 |= m2;
m1 &= m2;
m1 ^= m2;
```
Collections: Map

• Intended to replace "map-like" methods on arrays, which are now deprecated

```javascript
var D: domain(string);
var A: [D] int;
A["hello"] = 42; // warning: growing associative domains by assigning
                 // to an array is deprecated
```

• Can now be written as:

```javascript
var m = new map(string, int);
m["hello"] = 42;
```
Collections: Set

• Offers a lightweight alternative to domains
• Similar to both list and map, set operations can be made parallel-safe

```javascript
use Set;
var s1: set(int, parSafe=true);
coforall i in 1..8 with (ref s1) do
    s1.add(i);
writeln(s1 ^ s1);
```
Collections: Impact, Status

Impact:
- New list, set, and map data structures added to standard modules
- Vector-like and map-like methods on arrays are deprecated

Status:
- All three types have testing coverage
- List and map are used in the Mason, UnitTest, and TOML packages
Collections: Next Steps

**Next Steps:**

- Collect user feedback
- Investigate the performance of list and map
- Explore potential optimizations
- Make sure parallel safety strategies are appropriate
- Consider alternative underlying implementations
UnitTest
UnitTest: Background

• Existing options for writing Chapel user code tests are problematic
  • start_test
    • Not ideal for writing unit tests
    • Not a user-facing test feature, not supported by mason
  • mason test
    • Very limited capabilities by design
    • Only uses exit code of program to determine pass/fail status
UnitTest: This Effort

• Introduced a ‘UnitTest’ package module for writing unit tests in Chapel
• Implemented as a Google Summer of Code project

<table>
<thead>
<tr>
<th>Student</th>
<th>Mentors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Krishna Kumar Dey</td>
<td>Ben Albrecht, Lydia Duncan, Sam Partee</td>
</tr>
</tbody>
</table>

• Functions are designated as tests through their signature
  • Test functions must take a ‘borrowed Test’, throw, and return nothing:

  \[
  \textbf{proc} \ \textit{someTest}(t: \texttt{borrowed} \ \textit{Test}) \ \textbf{throws}
  \]
UnitTest: This Effort

- ‘mason test’ acts as the test runner both inside and outside of mason packages
  - Inside mason packages, runs tests in ‘test/’ directory
  - Outside mason packages, finds tests in current directory, unless path given
- Test runner compiles and runs tests
  - This allows resuming test suite after a halt
UnitTest: Test Assertions

• The ‘Test’ type provides JUnit-style assertion methods

```java
proc testFoo(t: borrowed Test) throws {
    t.assertTrue(FileSystem.isFile('foo'));
    t.assertFalse(FileSystem.isDir('foo'));
    t.assertEqual(here.id, 0);
    t.assertNotEqual(here.id, 1);
    t.assertGreaterthan(10, 1);
    t.assertLessThan(1, 10);
}
```
UnitTest: Example Usage

```chapel
use UnitTest;

proc testUpper(test: borrowed Test) throws {
    test.assertEqual('foo'.toUpperCase(), 'FOO');
}
UnitTest.main();

> mason test example.chpl
Ran 1 test in 11.3092 seconds

OK (passed = 1 )
```
UnitTest: Skipping Tests

• Metadata is attached to test functions via ‘Test’ methods
  • Tests can be skipped with or without a condition:

```plaintext
proc skip(reason: string) throws
proc skipIf(condition: bool, reason: string) throws
```
proc testFoo(t: borrowed Test) throws {
    t.skipIf(CHPL_LLVM == 'none', "This test requires LLVM");
    ...
}

proc testFoo(t: borrowed Test) throws {
    t.skip("This is not yet expected to work");
    ...
}
UnitTest: Test Dependencies

• Cross-test dependencies can be specified:
  • Functions are specified as first-class functions

```java
proc dependsOn(tests: argType ...?n) throws
```
[UnitTest: Test Dependency Example]

```rust
crate-

use FileSystem, UnitTest;

proc testMoveDir(t: borrowed Test) throws {
    // testMkdir must be run before testMoveDir is run
    t.dependsOn(testMkdir);
    moveDir('foo', 'bar');
    t.assertTrue(isDir('bar'));
}

proc testMkdir(t: borrowed Test) throws {
    mkdir('foo');
    t.assertTrue(isDir('foo'));
}
```
UnitTest: Multilocation Tests

• Tests can specify how many locales they require
  • Tests can list discrete numbers of locales supported:
    
    \[
    \text{proc addNumLocales(locales: int ...?n) throws}
    \]

• Tests can provide a range of locales supported:

  \[
  \text{proc maxLocales(value: int) throws}
  \]

  \[
  \text{proc minLocales(value: int) throws}
  \]

• The test runner makes multiple passes with different numbers of locales ...
  ... such that each tests is run with a number of locales that it requires

• Running one test with \( N \) numLocales values requires wrapping it with \( N \) tests
proc foo(t: borrowed Test) throws {
    t.addNumLocales(4);
    // Will run with 4 locales only
}

proc bar(t: borrowed Test) throws {
    t.minLocales(8);
    t.maxLocales(16);
    // Will run with 16 locales only (defaults to max of locale bounds)
}
UnitTest: Multilocal Test Example 2

```plaintext
proc foo(t: borrowed Test) throws {
    t.addNumLocales(4, 8);
    // Will run with 8 locales only
}

proc bar(t: borrowed Test) throws {
    t.minLocales(8);
    t.maxLocales(16);
    // Will run with 8 locales only
}
```
proc foo() {
    // Will run with 4 and 8 locales, because called from foo4 and foo8
}

proc foo4(t: borrowed Test) throws {
    t.addNumLocales(4);
    foo();
}

proc foo8(t: borrowed Test) throws {
    t.addNumLocales(8);
    foo();
}
UnitTest: Status, Next Steps

Status:
• Chapel has an official user-facing test framework
• UnitTest is available as a package module
• ‘mason test’ is the UnitTest test runner

Next Steps:
• Support test suites as modules, records, and/or classes
• Support introspective assertions, similar to pytest
  • Allows a general assert function to give useful information when failing
• Respond to user feedback
EpochManager
EpochManager: Background

• Would like Chapel to include a library of lock-free data structures:
  • After all, Chapel has an emphasis on productivity and performance
• Two missing features preventing implementation of such structures
  1. Compare-and-swap is not implemented for Chapel classes
     • Challenge: 128-bit wide pointers in multilocusle configurations
  2. A general solution to the ABA problem is needed
     • Challenge: Chapel is not garbage collected
     • Challenge: Generation counters would require 256-bit compare-and-swap
     • Challenge: Hazard Pointers are not sufficiently composable
This Effort: Add prototype EpochManager and related package modules

- Implemented as a Google Summer of Code project
- EpochManager solves the ABA problem by deferring deletion in Epochs
- AtomicObjects provides compare-and-swap on local or wide pointers
- These enabled implementation of LockFreeQueue and LockFreeStack

Next Steps:

- Migrate 128-bit CAS support to runtime
- Handle nested usage of EpochManager

<table>
<thead>
<tr>
<th>Student</th>
<th>Mentors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garvit Dewan</td>
<td>Louis Jenkins</td>
</tr>
<tr>
<td></td>
<td>Michael Ferguson</td>
</tr>
</tbody>
</table>
Module Improvements

- Sort
- Reflection
- Linear Algebra
Sort Improvements
Sort: Background, This Effort

**Background:** A parallel radix sort was added in 1.19
- Only recursive sorts were run in parallel
  - count and bucketize were serial
- Not as fast as some C++ sort implementations
- No support for distributed sorting

**This Effort:**
- Improve the Sort module by fixing bugs and improving performance
- Prototype more performant algorithms and distributed sorting
Sort: Implementation Improvements

• For radix sort:
  • count step is now parallel
  • now supports floating point numbers as well as 'c_string'
• Arrays with domains using ranges with 'align' can now be sorted
• isSorted() is now efficient for Block-distributed arrays
Sort: Prototype Sorts

- Developed prototype two-array radix sort and distributed sort
  - These are included as undocumented features in the Sort module
- Two-array radix sort can offer significantly better performance
  - ~5x faster than the recursive implementation
  - but using 2x the space
- Distributed radix sort is functional but has scaling problems
- Described in a [CHIUW 2019 presentation](#)
Sort: Next Steps

**Next Steps:** Follow up on investigations to improve Sort module

- Decide if 2x space overhead is acceptable
- Investigate in-place parallel algorithms
- Resolve performance problems with distributed sort
- Investigate other distributed sort algorithms
Reflection
Improvements
Reflection: Background, This Effort

**Background:** There was no way to query the location of the current code

- Users desired a way to know things like file name and line number
  - e.g. for logging

**This Effort:** Added source location query functions to the Reflection module

```plaintext
proc getLineNumber() param : int
proc getFileName() param : string
proc getRoutineName() param : string
proc getModuleName() param : string
```
Impact: Chapel programs can now query source locations

```plaintext
use Reflection;

writeln(getFileName(), ":", getLineNumber(), ", ",
        getModuleName(), ".", getRoutineName());
```

Next Steps:

- Add functionality for nested routines and modules
  - 'get[Module/Routine]Name()' only return the nearest module/routine
  - Want to be able reason about the full stack in nested cases
- Enable getting location information where a routine is called
Linear Algebra Improvements
LinearAlgebra Module: Background, This Effort

**Background:** Chapel’s LinearAlgebra module provides linear algebra routines

**This Effort:** New routines and improved support for distributed and/or sparse data

• New routines for CSR and COO matrices:

```cpp
proc isDiag(A): bool;
proc isHermitian(A): bool;
proc isSymmetric(A): bool;
```
LinearAlgebra Module: This Effort

- The following are implemented as part of Google Summer of Code

<table>
<thead>
<tr>
<th>Student</th>
<th>Mentors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alvis Wong</td>
<td>Engin Kayraklioglu, Ben Albrecht</td>
</tr>
</tbody>
</table>

- New routines for linear system solution:
  ```
  proc lu(A);
  proc solve(A, b);
  ```

- Miscellaneous routines:
  ```
  proc setDiag(X, offset=0, val=0);
  proc det(A);
  proc inv(A);
  proc jacobi(A, ref x, b, tol=0.0001, maxiter=1000);
  ```

- More support for distributed vectors/matrices: 'plus', 'minus', faster 'dot'
LinearAlgebra Module: Impact, Next Steps

**Impact:**
- Support for more routines, on different types of data (sparse or distributed)
- Improved dot performance

**Next Steps:** Continue improvements
- More support for distributed and/or sparse data
- GPU support
For More Information

For a more complete list of library-related changes in the 1.20 release, refer to the following sections of the CHANGES.md file:

• Deprecated / Removed Library Features
• Standard Library Modules
• Package Modules
• Bug Fixes
This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries (“Hewlett Packard Enterprise”) may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties’ control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray’s business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise’s businesses; the competitive pressures faced by Hewlett Packard Enterprise’s businesses; risks associated with executing Hewlett Packard Enterprise’s strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise’s other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.
THANK YOU

QUESTIONS?

chapel_info@cray.com
@ChapelLanguage
chapel-lang.org

cray.com
@cray_inc
linkedin.com/company/cray-inc/-