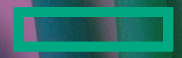


Hewlett Packard
Enterprise

Chapel 1.33 / 2.0 Release Notes: Library Improvements

Chapel Team

December 14, 2023 / March 21, 2024

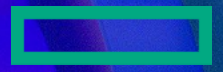
Outline

- IO Standard Library
- Parallel IO Library
- Zarr I/O Library
- Debugger.breakpoint
- Other Library Improvements

IO Standard Library

IO Standard Library

- Stabilization Changes
- Other IO Improvements



IO Stabilization

Background & This Effort

Background:

- The IO library has been the subject of many deprecations and changes in the march towards 2.0
- In the 1.32 release, the last of the major changes were finished
  - E.g., the serialization framework, formatted I/O improvements, and many deprecations

This Effort:

- Made some additional breaking changes after further consideration
- Fixed a handful of bugs
- Improved documentation
- Removed most deprecated features
- Added some unstable features

IO Stabilization Changes

Locking Default

Background, This Effort, and Next Steps

Background:

- The ‘locking’ property of fileReader/Writer defaulted to ‘true’ to provide parallel safety “out of the box”
- Over time, we found that users were surprised by this, and sometimes encountered performance issues

This Effort:

- Deprecated ‘locking’ default in 'file.reader' and 'file.writer' factory methods
  - Requires users to be explicit about what they want
- Deprecated ‘locking=true’ in favor of ‘false’ default in 'openReader' and 'openWriter'
  - Warns users that this value will change in an upcoming release
  - Can use '-sOpen[Reader|Writer]LockingDefault=false' to get future behavior
- Note: stdin/stdout/stderr remain locking by default

Next Steps:

- Remove deprecated defaults, and change to using ‘false’ in 'openReader'/'openWriter’
- Consider having ‘file.reader’/‘file.writer’ default to ‘locking=false’ as well

Changes to Binary Serializer

Background, This Effort, and Impact

Background:

- The 'binary[De]Serializer' types were initially implemented to use a “structured” binary format
  - Specifically, included a length integer for strings and a nil-byte for classes
- The intent was to make it easier to read serialized binary data compared to the old 'iokind' implementation
- Without a "raw" format it was difficult to use files generated by 'iokind' or external sources

This Effort:

- Changed the 'binary[De]Serializer' types to be “unstructured” by default in 1.33
  - Motivated by concerns over stability of the “structured” format
- Moved “structured” implementation into unstable 'ObjectSerialization' package module
  - Intended as a package that could eventually support more robust “pickling” of data, but very much a prototype today

Impact:

- Easier to port deprecated 'iokind' code to use 'binary[De]Serializer’ and to ingest unformatted data files

Renaming 'ioendian'

Background, This Effort, and Impact

Background:

- The 'ioendian' enum was used in various methods to indicate the desired endianness in binary I/O
- Most other types with an 'io' prefix were renamed as part of module stabilization

This Effort:

- Deprecated 'ioendian' and renamed to 'endianness'

Impact:

- Improved consistency across the IO standard library's naming

Other IO Improvements

IO Stabilization — Bug Fixes

This Effort

- Made fixes to remote I/O
  - Correctly handle remote strings in 'readTo', 'readThrough', and 'readLine'
  - Fixed a bug with the 'seek' method on remote files
  - Fixed a bug when writing arrays in binary across locales

- Binary bugs
  - Correctly throw an 'UnexpectedEofError' for 'readBinary'
  - Correctly return 'false' from 'fileReader.read' when using 'binary[De]Serializer'
  - Improved read/writeBinary performance for little and big endian

IO Stabilization — Documentation Improvements

Background, This Effort, and Impact

Background:

- The size of the IO library and recent rate of change has occasionally led to gaps in documentation quality
- Some useful documentation was also unintuitively split into the ChapelIO module

This Effort:

- Performed a full pass over the IO and FormattedIO modules with dozens of improvements
- Merged the ChapelIO documentation into the IO module's documentation

Impact:

- Documentation is clearer and more accurate than ever
- Users don't need to look at separate documentation page (i.e., ChapelIO) to find things like top-level 'writeln'

IO Stabilization — Removed Features

Background and This Effort

Background:

- Numerous features have been deprecated in past releases as a part of the IO stabilization effort

This Effort:

- Identified and removed many features that have been deprecated for at least two releases
- Formatted IO: '%t', '%jt', '%ht' generic specifiers
- On both fileReader/Writer:
  - The '.binary' and '.writing' properties
  - The 'advancePastBytes' method
- Removed the 'file.lines' method
- Removed the 'iostyle' and 'iokind' types, along with corresponding ‘style’ and ‘kind’ arguments to methods
- Removed support for readThis/writeThis methods
- See CHANGES.md for a full list

IO Stabilization — New Unstable Features

This Effort

- Added unstable 'openStringReader' routine to enable reading from a string:

```
var r = openStringReader("hello world!\nI'm a string!");
writeln(r.readLine());   // hello world!
writeln(r.readLine());   // I'm a string!
```

- Added unstable 'getFile' method to 'fileReader' and 'fileWriter'
  - Returns the underlying 'file'

- Added 'IOSkipBufferingForLargeOps' config param to control dynamic buffering optimization
  - On by default; compile with '-sIOSkipBufferingForLargeOps=false' to disable

Parallel IO Library

Parallel IO Library

Background and This Effort

Background:

- The IO library has the necessary building blocks for implementing parallel and distributed I/O operations
  - however, it didn't have higher-level abstractions for parallel IO
- Reading files with variable-length items in parallel is tricky
  - users may not want to implement this kind of thing on their own

This Effort:

- Created a 'ParallelIO' package module with several abstractions for reading files in parallel
  - parallel iteration over lines (or delimited items):

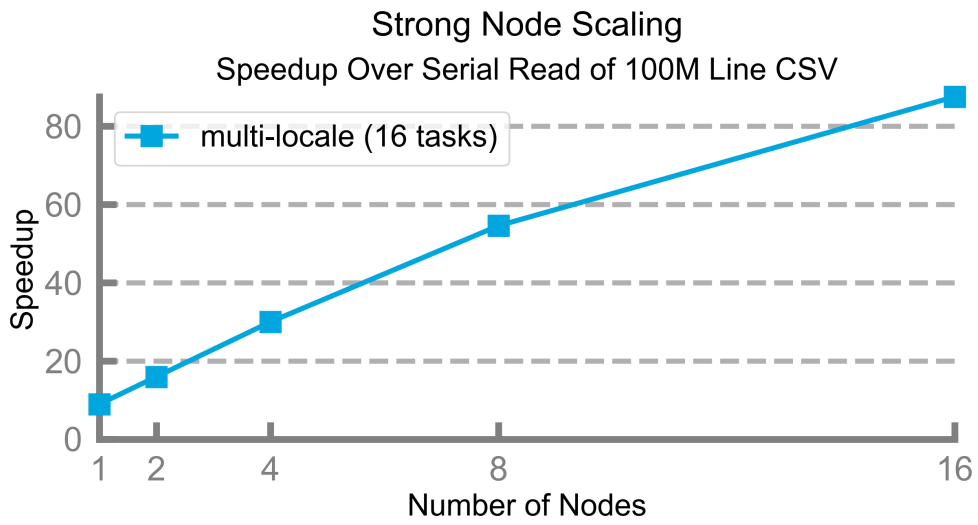
```
forall line in readLines("file.txt") {
  ...
}
```

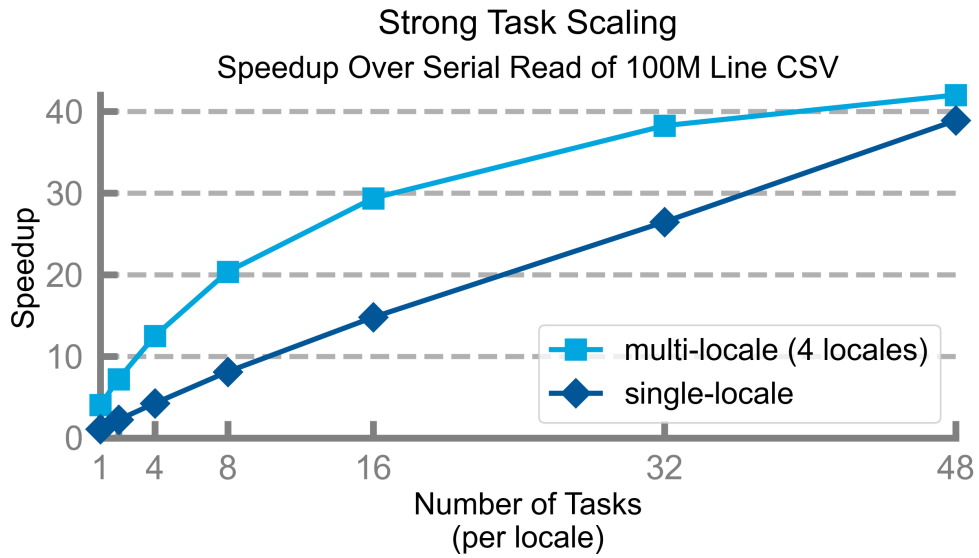
  - reading lines or delimited items into an array (default or block-distributed):

```
var arr = readDelimitedAsArray("data.csv", t=myDataType, delim="\n");
```

Parallel IO Library
Impact

- The module provides scalable distributed reads and significant speedups over reading serially
- Speedup for reading a 100M-row CSV file into an array of records (takes 10 seconds serially):





collected on a Cray XC: 48-core Cascade Lake CPUs, Lustre filesystem, Aries network

Parallel IO Library

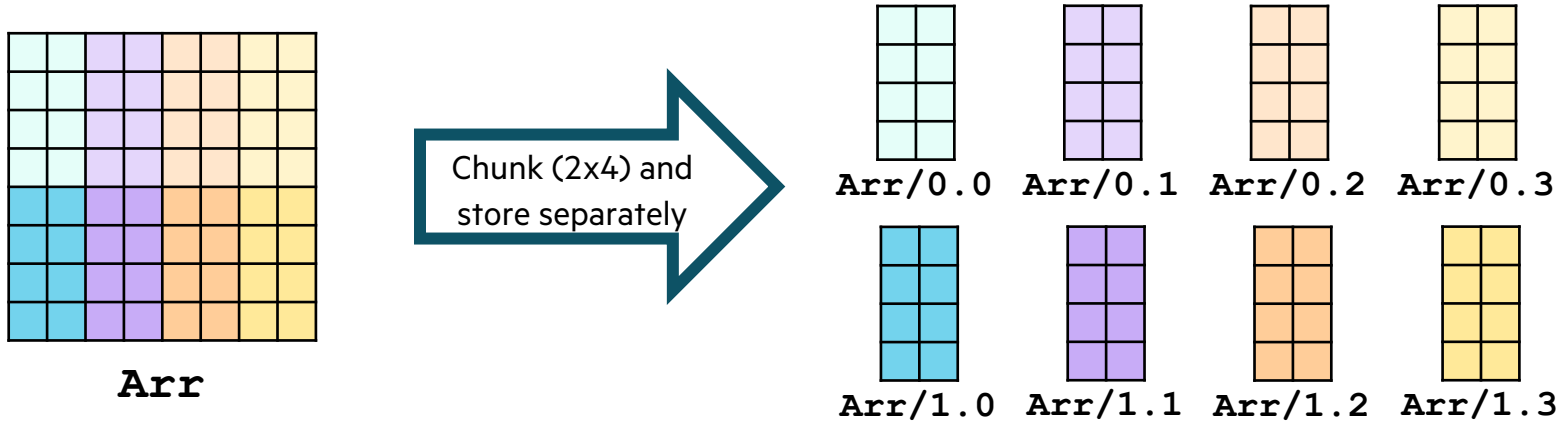
Next Steps

- Improve iterator support:
  - add leader/follower versions for zippered iteration
  - add multi-locale versions
- Improve error handling:
  - fall back to serial IO if parallelization isn't possible
- Investigate updating Arkouda's CSV support to use ParallelIO
- Consider moving some ParallelIO functionality into the standard IO library
  - planning to move at least the 'readLines' iterator to IO
- Add support for writing files in parallel


Zarr I/O Library

Zarr I/O Library
Background

- The Zarr File Format
  - Multi-dimensional data, broken into chunks, compressed, and stored separately
  - Commonly used for geospatial data with a time dimension



- Support was requested by users
  - [C]Worthy is a startup doing ocean modeling to support carbon sequestration
  - They are developing a Chapel application that is bottlenecked by reading/writing different Zarr stores

Zarr I/O Library
This Effort

- Added an unstable Chapel library for distributed parallel Zarr I/O
  - Supports reading/writing stores on a local file system, e.g.,

```
use Zarr;

// Read a Zarr array from the specified location, indicating the array's element type and dimension
var arr = readZarrArray("path/to/zarr/store", int, dimCount=2);

// Write an array into the Zarr format, specifying the location to write it to, and the shape of the chunk to write
// Since this is a 2-dimensional array, we need to specify 2 dimensions for the chunk shape
writeZarrArray("path/to/intended/dest", arr, chunkShape=(dim1, dim2));
```

Zarr I/O Library

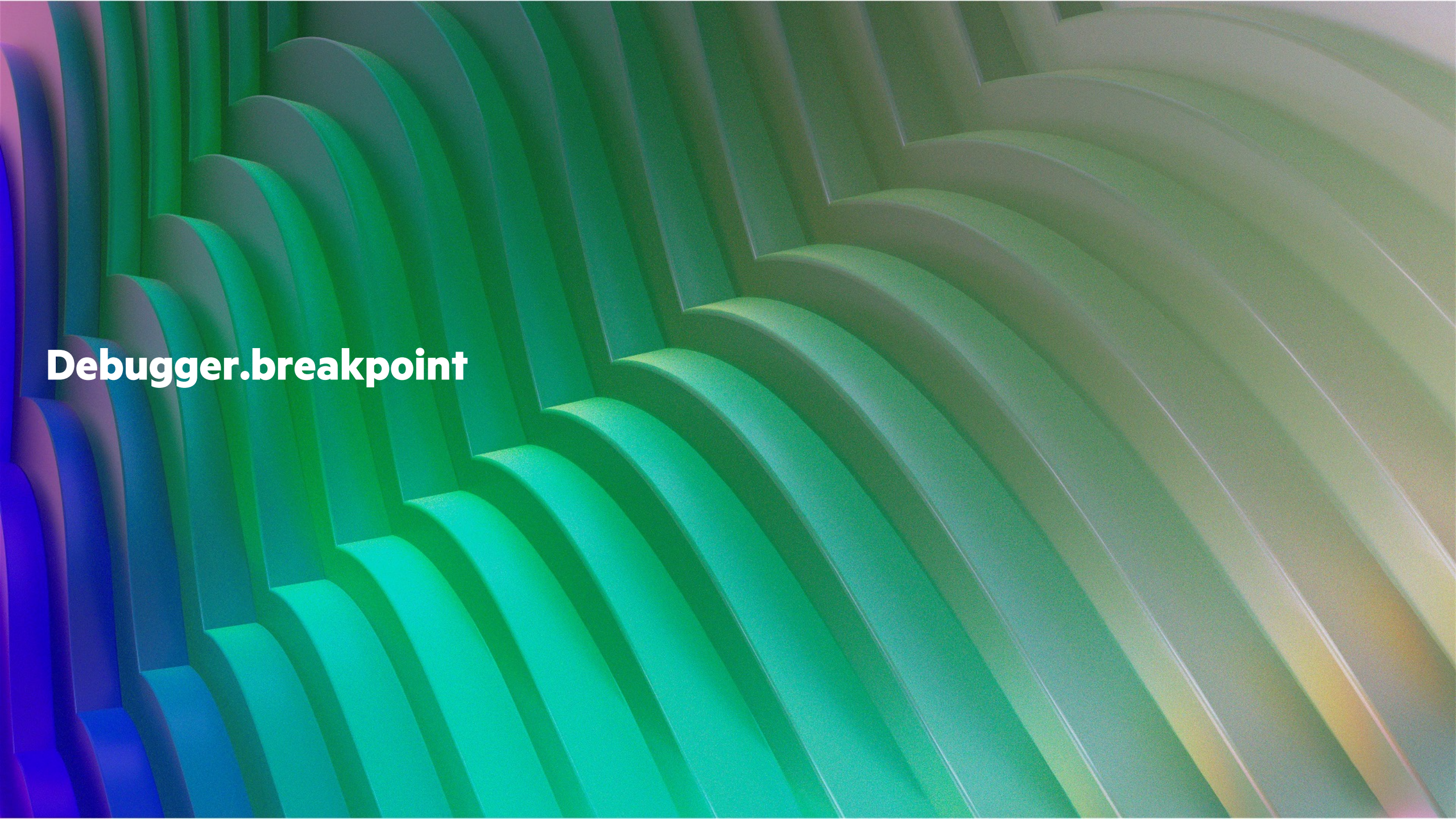
Status and Next Steps

Status:

- [C]Worthy team integrating library into their modeling code

Next Steps:

- Continue development of library based on [C]Worthy user needs
  - Support cloud-based Zarr stores (S3, Google Cloud, Azure), where important climate datasets are stored
  - Support array “groups”, which are used to combine multiple arrays into a single dataset
- User request: Integrate into Arkouda using emerging multi-dimensional array support


Debugger.breakpoint

Debugger.breakpoint

This Effort:

- Added a new unstable library, ‘Debugger’
- Supports users writing explicit breakpoints in code

```
import Debugger;

for i in 1..10 {
  writeln("i is ", i);
  Debugger.breakpoint;
}
```

- This pauses execution when running within GDB or LLDB
  - works exactly like a user-defined breakpoint in the debugger
  - only enabled when compiled with ‘-g’
  - implemented with debugger interrupts

Next Steps: stabilize the library

Other Library Improvements

Other Library Improvements

For a more complete list of library changes and improvements in the 1.33 and 2.0 releases, refer to the following sections in the CHANGES.md file:

- New Standard Library Features (2.0) / Standard Library Modules (1.33)
- New Package Module Features (2.0) / Package Modules (1.33)
- Changes/Feature Improvements in Libraries
- Name Changes in Libraries
- Deprecated/Unstable/Removed Library Features
- Performance Optimizations / Improvements
- Documentation Improvements
- Bug Fixes for Libraries
- Developer-oriented changes: Module changes
- Developer-oriented changes: Testing System


Thank you
https://chapel-lang.org
@ChapelLanguage