Hewlett Packard
Enterprise

Chapel 2.1 / 2.2 Release Notes: Language / Library Improvements

Chapel Team

June 27, 2024 / September 26, 2024

Outline

- Remote Variable Declarations
- Sort Module Stabilization
- Random Module Improvements
- Sparse Improvements
- Custom Allocators
- I/O Improvements
- Image Module
- Other Language/Library Improvements

Remote Variable Declarations

Remote Variable Declarations

Background

- Variables are stored on the locale where their declaration executes
- ‘on’ statements are used to transfer task execution to a different locale
- A common pattern is to allocate an array on one locale, but continue execution on another locale
  - Historically, this has required two nested ‘on’ statements, since each block creates a new scope

```
writeln("on initial locale");
on here.gpus[0] {
  var GpuA = foreach i in 1..128 do i*i;
  on Locales[0] {
    var CpuB = GpuA;
    writeln("copied array back onto CPU, its value is ", CpuB);
  }
}
```

- Specification describes “remote variable declarations”, which do not introduce new scopes like ‘on’ does

```
on here.gpus[0] var GpuA = foreach i in 1..128 do i*i;
```

  - Yet these had never been implemented

Remote Variable Declarations

This Effort

In Chapel 2.1:

- Added support for remote declarations of single variables
- Only when the initialization expression was a loop or promoted expression, was it executed on the target locale
- The type/initialization were restricted to be of a matching type

```
writeln("on initial locale");
on here.gpus[0] var GpuA = foreach i in 1..128 do i*i;
var CpuB = GpuA;
writeln("copied array back onto CPU, its value is ", CpuB);
```

In Chapel 2.2:

- Ensured that initialization expressions are always executed on the target locale
- Added support for multi-declarations and type coercions

```
on Locales.last var A: [1..10] int = 2,
                    B = foreach i in 1..10 do i*i;
```

Remote Variable Declarations

Status and Next Steps

Status:

- Remote variables are operational
- They are currently marked as unstable
- Some minor performance and semantic issues remain
  - Remotely allocating classes causes two allocations (one for remote variable storage), but could use only one
  - Multi-variable remote declarations invoke a remote task for each variable
  - Remote variables are not supported as fields of records or classes

Next Steps:

- Stabilize remote variables
- Resolve the semantic issues listed above
  - Multi-variable declarations are of particular interest to allocate efficiently, avoiding remote execution overhead

‘Sort’ Module Stabilization

‘Sort’ Module Stabilization

Background and This Effort

Background:

- ‘Sort’ module was one of the higher priority unstable modules
  - Sorting is taught in basic programming courses
  - Generally useful for a wide variety of applications

This Effort:

- Reviewed and stabilized most of the documented interface
  - Deprecated ‘reverseComparator’ and ‘defaultComparator’ module-scoped variables
  - Cleaned up the argument lists for ‘sort( )’, ‘isSorted( )’ and ‘iter sorted( )’
  - Replaced ‘list.sort( )’ with ‘sort(x: list)’, for consistency
  - Removed many undocumented submodules

‘Sort’ Module Stabilization

This Effort (New Features)

- Defined ‘keyComparator’/‘keyPartComparator’/‘relativeComparator’ interfaces for defining comparators

```
record myComparator: keyComparator {
  proc key(x) do return x.fieldA;
}

var arr: [0..9] myType = …;
sort(arr, new myComparator());
```

```
record myType {
  var fieldA: int;
  …
}
```

- Enabled support for stable-value sorting via ‘sort(…, stable=true)’

- Added support for sorting with a region argument

```
var arr = [-2, 6, 11, 1, 5, -5, 8, 7];
sort(arr, new DefaultComparator(), 2..7);   // arr = [-2, 6, -5, 1, 5, 7, 8, 11]
```

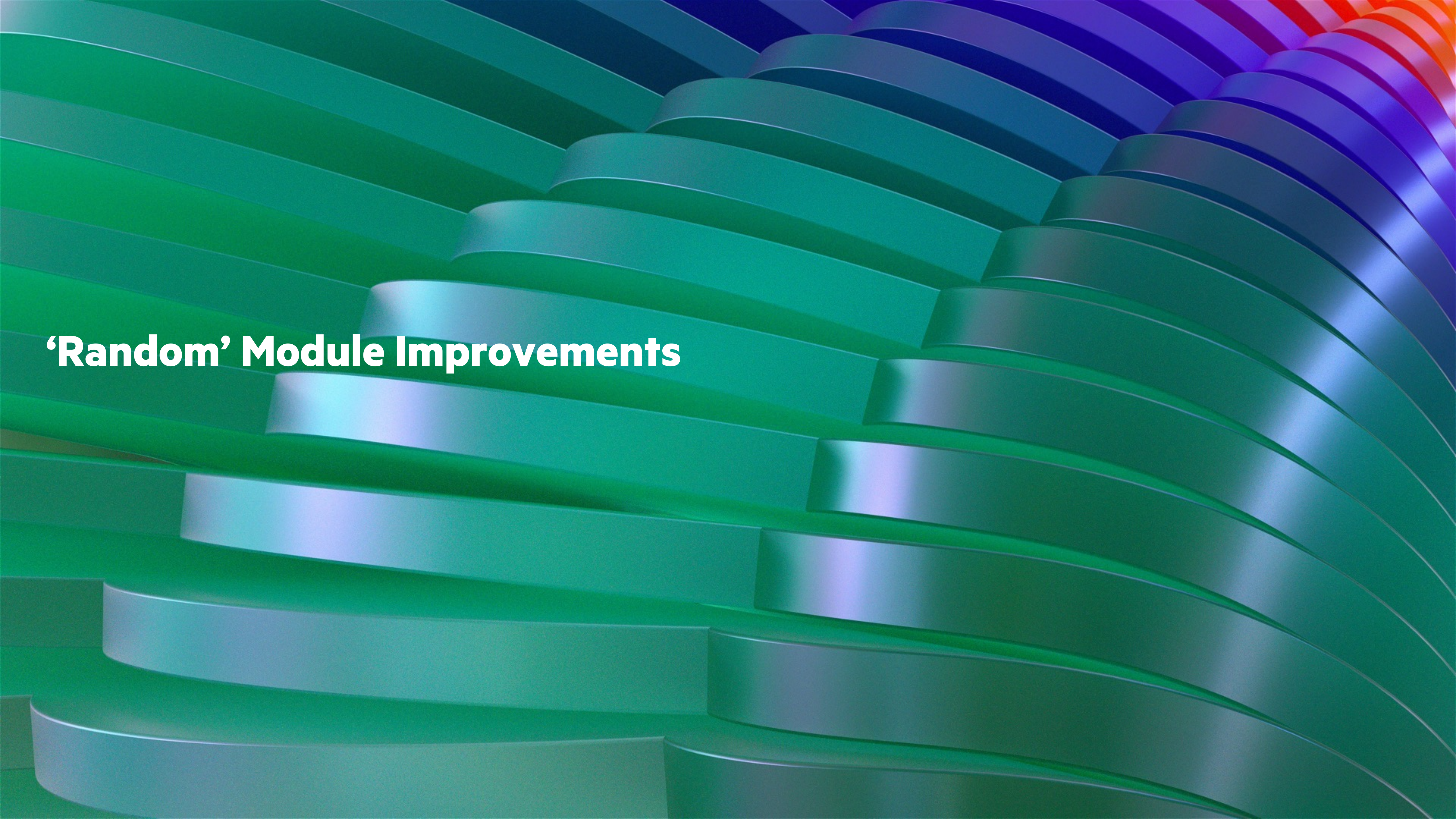
‘Sort’ Module Stabilization
Impact and Next Steps

Impact:

- Most of ‘Sort’ module interface is now stable!
  - Users don’t need to worry about ‘isSorted( )’, ‘sorted( )’ or most versions of ‘sort( )’ changing underneath them
  - ‘sort( )’ with a region argument, the new interfaces, & the names for ‘DefaultComparator’/’ReverseComparator’ are unstable
- Users now have clearer blueprint for writing their own comparators
- ‘use Sort’ adds less clutter to the user’s namespace due to undocumented Sort submodules
- Sorting is more unified across types

Next Steps:

- Rename ‘DefaultComparator’ and ‘ReverseComparator’ types
  - To match Standard Module Style Guide for records
  - Wasn’t possible earlier due to module-scoped instances using the intended name
- Enable ‘sort( )’ on distributed arrays



‘Random’ Module Improvements

‘Random’ Module Improvements

This Effort:

- implemented weighted random sampling

```
writeln(sample([1, 2, 3], n=10, weights=[0.1, 0.1, 0.8], withReplacement=true));
// prints: 3 1 3 3 1 3 3 2 3 3
```

- added multi-dimensional support to several procedures
  - ‘shuffle( )’, ‘permute( )’, ‘choose( )’, ‘sample( )’

```
const x = reshape([1, 2, 3, 4, 5, 6], {1..2, 1..3});
writeln(permute(x));
// prints: 3 6 4
           2 1 5
```

Impact:

- new features were useful for aligning Arkouda's random module with NumPy

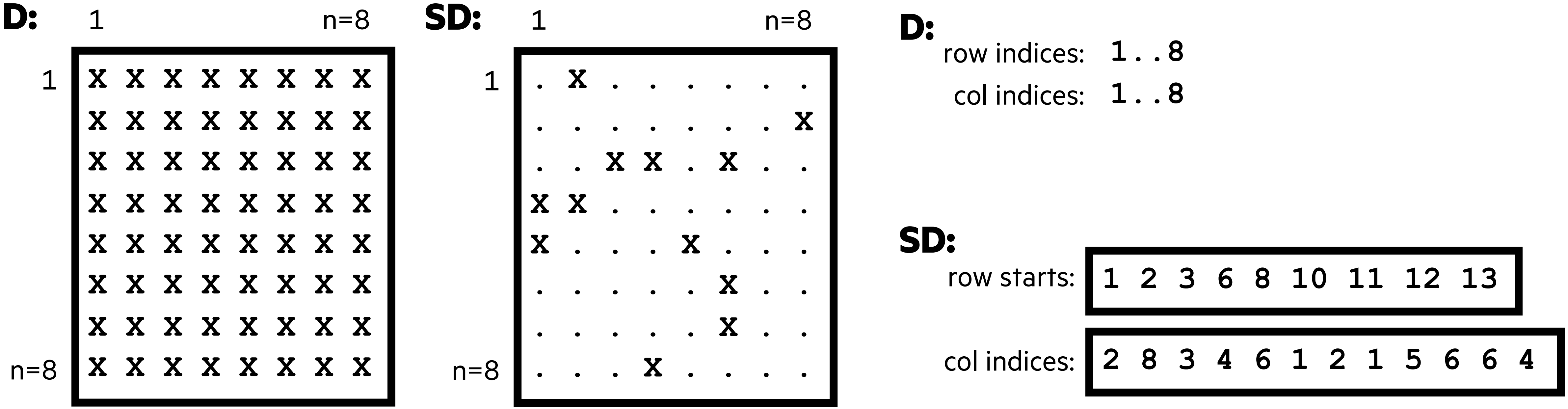
Sparse Improvements

Sparse Improvements

Background: Local Domains

- CSR/CSC are common 2D sparse matrix representations, supported by Chapel's 'LayoutCS' module

```
const D = {1..n, 1..n},
      SD: sparse subdomain(D) dmapped new dmap(new CS(compressRows=true)) = …;
```



D:

```
      1             n=8
  1   x x x x x x x x
      x x x x x x x x
      x x x x x x x x
      x x x x x x x x
      x x x x x x x x
      x x x x x x x x
      x x x x x x x x
n=8   x x x x x x x x
```

SD:

```
      1             n=8
  1   . x . . . . . .
      . . . . . . . x
      . . x x . x . .
      x x . . . . . .
      x . . . x . . .
      . . . . . x . .
      . . . . . x . .
n=8   . . . x . . . .
```

D:

row indices: `1..8`

col indices: `1..8`

SD:

row starts: `1 2 3 6 8 10 11 12 13`

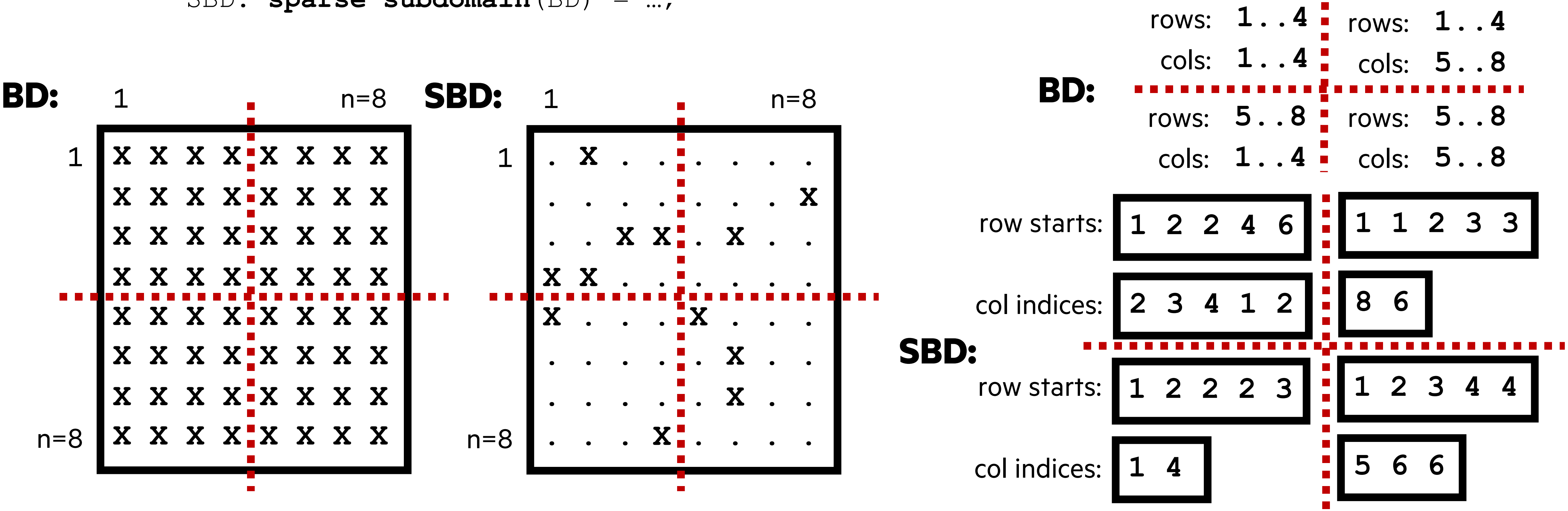
col indices: `2 8 3 4 6 1 2 1 5 6 6 4`

Sparse Improvements

Background: Distributed Domains

- CSR/CSC can also be combined with Block-distributed sparse arrays to specify a per-locale sparse layout

```
const D = {1..n, 1..n},
      BD = D dmapped new blockDist(D, sparseLayoutType=CS(compressRows=true)),
      SBD: sparse subdomain(BD) = …;
```



BD:

```
     1             n=8
  1  X X X X | X X X X
     X X X X | X X X X
     X X X X | X X X X
     X X X X | X X X X
     --------+--------
     X X X X | X X X X
     X X X X | X X X X
     X X X X | X X X X
n=8  X X X X | X X X X
```

SBD:

```
     1             n=8
  1  . X . . | . . . .
     . . . . | . . . X
     . . X X | . X . .
     X X . . | . . . .
     --------+--------
     X . . . | X . . .
     . . . . | . X . .
     . . . . | . X . .
n=8  . . . X | . . . .
```

BD:

```
rows: 1..4   |  rows: 1..4
cols: 1..4   |  cols: 5..8
-------------+-------------
rows: 5..8   |  rows: 5..8
cols: 1..4   |  cols: 5..8
```

SBD:

```
row starts:  1 2 2 4 6  |  1 1 2 3 3
col indices: 2 3 4 1 2  |  8 6
------------------------+------------
row starts:  1 2 2 2 3  |  1 2 3 4 4
col indices: 1 4        |  5 6 6
```

Sparse Improvements

More Background and This Effort

Background:

- Chapel’s sparse features are unstable and in need of improvement
  - For example, the declarations shown previously should be simplified to improve readability / comprehension
  - In other cases, capabilities are missing, preventing the expression of important patterns

This Effort:

- Implemented some of these missing capabilities:
  - Methods for traversing CSR/CSC arrays
  - Procedures for getting or setting a locale’s sub-domain/sub-array of a block-distributed sparse domain/array
  - Support for querying a sparse array’s target locales
  - Support for copying between sparse arrays

Sparse Improvements

This Effort: Iteration Improvements

- Added '.rows( )'/'.cols( )' queries to get the dense rows/columns of a CSR/CSC domain or array
  - These are essentially 2D-specific sugar for '.dim(0)'/'.dim(1)'

- Added serial iterators to yield the indices and values of a CSR/CSC array in a given row/col

```
for (col, val) in MySpsArr.colsAndVals(row) do …   // yields column indices and values for CSR arrays
for (row, val) in MySpsArr.rowsAndVals(col) do …   // yields row indices and values for CSC arrays
```

- This example uses both methods to traverse a CSR array in parallel:

```
const D = {1..n, 1..n},
      SD: sparse subdomain(D) dmapped new dmap(new CS(compressRows=true)) = …;

var A: [SD] real = …;

forall r in A.rows() do
  for (c, a) in A.colsAndVals(r) do
    writeln("A[", (r,c), "] = ", a);
```

Sparse Improvements

This Effort: Local Block-Sparse Setters/Getters

- Added the ability to query and set the local indices/elements of a sparse, block-distributed domain/array
  - Permits an algorithm to query and/or replace a locale’s local block of sparse indices / elements

```
var localSparseDomain = …
    localSparseArray: [localSparseDomain] real;

MyBlockSparseDomain.setLocalSubdomain(localSparseDomain);
MyBlockSparseArray.setLocalSubarray(localSparseArray);


…

const localInds = MyBlockSparseDomainOrArray.localSubdomain(),   // pre-existing query
      localData = MyBlockSparseDomain.getLocalSubarray(targetLocRow, targetLocCol);
```

Sparse Improvements

This Effort: Orthogonality Improvements

- Added features which are already standard for dense domains / arrays:
  - The ability to copy between sparse arrays with distinct-but-equivalent domains:

```
const D = MySpsArr.domain,
      A: [D] MySpsArr.eltType = MySpsArr;   // was: an error about being unable to zip arbitrary sparse arrays
                                            // now: works
```

  - The ability to query the target locales over which a block-sparse array is distributed:

```
coforall loc in MyBlockSparseArray.targetLocales() do
  on loc do
    writeln("locale ", loc, " owns: ", MyBlockSparseArray.localSubdomain());
```

Sparse Improvements

Impact and Status

Impact:

- With current features, a CSC x CSR matrix-matrix multiplication algorithm can now be written cleanly
  - Local or block-distributed versions

Status:

- Sparse domains and arrays are much improved as of Chapel 2.1

Sparse Improvements

Next Steps

- Update implementation and naming of CSR/CSC layouts:
  - Convert 'CS' from a class to a record, as standard distributions were in Chapel 1.32
  - Split into two layout types for clarity: 'csrLayout'/'cscLayout'
  - Rename module for clarity and consistency with distributions
- Continue improving sparse iterators
  - Parallelize row/col + val iterators
  - Extend CSR/CSC row-/col-specific iterators to domains
  - Support whole-domain / array parallel iterators
  - Consider supporting 'sparseRows( )'/'sparseCols( )' iterators that only yield non-empty row/column indices
- Improve naming and symmetry in block-sparse per-locale set/get procedures
- Continue to improve sparse features
  - Study and tune performance
  - Continue to identify other missing methods and features
  - Work toward stabilizing sparse domains and arrays

Custom Allocators

Custom Allocators

Background

- Heap objects are allocated semi-transparently in Chapel
  - Explicit allocation occurs with ‘new’ on classes
  - A user may be unaware that heap allocation occurs with some types (e.g. arrays, domains, strings, etc.)

- Users have some control over where in the program these allocations occur

```
class MyClass { var field: int; }
var x = new unmanaged MyClass(1);   // x is a pointer to the heap

var arr = [1, 2, 3, 4, 5];          // arr is on the heap

...

delete x;   // x is unmanaged and must be deleted to free the memory
            // arr’s memory is managed automatically
```

Custom Allocators

This Effort

- Added support for defining custom allocators

```
use Allocators, CTypes;
record myPool: allocator {    // 'allocator' is an interface
  var memoryChunk: c_ptr(void);
  proc ref allocate(n: int): c_ptr(void) { … }
  proc ref deallocate(p: c_ptr(void)) { … }
}
```

- Users can use custom memory allocators to create classes

```
var pool = new myPool();

class MyClass { var field: int; }
var x = newWithAllocator(pool, unmanaged MyClass, 1);   // allocate memory for x using 'pool.allocate( )'
...
deleteWithAllocator(x);                                   // deallocate x's memory using 'pool.deallocate( )'
```

Custom Allocators

Impact and Next Steps

Impact:

- Gives users finer control over memory allocations for performance-critical situations
- 2x–4x improvement on binary-trees benchmark (system-dependent)
  - The benchmark allocates and deallocates large binary trees
  - Performance gained by using a bump allocator to perform bulk memory operations

binary-trees version	Time (s)	Speedup (Cumulative)
Previous fastest June 2024	3.65	N/A
Inner loop parallel	1.88	1.94x
Inner loop parallel with bump allocator	0.85	4.29x

Next Steps:

- Expand support for allocators to include other heap objects (e.g., arrays)
- Add first-class language support for using allocators

I/O Improvements

I/O Improvements

This Effort:

- Added parallel & distributed versions of the 'fileReader.lines( )' iterator

```
forall line in openReader("data.txt").lines(targetLocales=Locales)
  do write("on locale ", here.id, " read: ", line);
```

  - Also added multi-locale support to iterators in 'ParallelIO' module

- Added a default value of ‘false’ to the 'locking' argument in 'openReader( )' and 'openWriter( )'
  - Makes the common, higher-performance mode the default
  - 'stdin', 'stdout' and 'stderr' still lock to facilitate safe accesses from multiple threads

- Added 'toJson( )' and 'fromJson( )' helpers for (de)serializing 'string' values in JSON format

```
use JSON, List;
record R { var x: real; var y: list(int); }

const myR = fromJson('{"x": 3.14, "y": [1, 2, 3]}', R);
writeln(toJson(myR));    // prints: {"x":3.140000e+00, "y":[1, 2, 3]}
```

I/O Improvements

This Effort:

- Added new ‘precisionSerializer’ for specifying padding and precision of all numerical values

```
use PrecisionSerializer;
const arr = [1.123456789, 2.123456789, 3.123456789, 4.123456789],
      fourPaddedDigits = new precisionSerializer(precision=3, padding=9);

stdout.withSerializer(fourPaddedDigits).writeln(arr);
// prints: "    1.123    2.123    3.123    4.123"
```

- Significantly improved the performance of procedures that read data into ‘string’ or ‘bytes’:
  - ‘readAll( )’, ‘readString( )’, ‘readBytes( )’, ‘readBinary( )’

- Optimized away string copies in the ‘regex.replace( )’ method
  - Observed a 20%–25% performance improvement for the ‘regex-redux’ benchmarks

‘Image’ Module

‘Image’ Module

This Effort

- Added an unstable ‘Image’ module

- Supports reading and writing PNG, JPEG, and BMP images
- Some support for creating MP4 videos
  - The ‘mediaPipe’ type provides a nice wrapper around ‘ffmpeg’

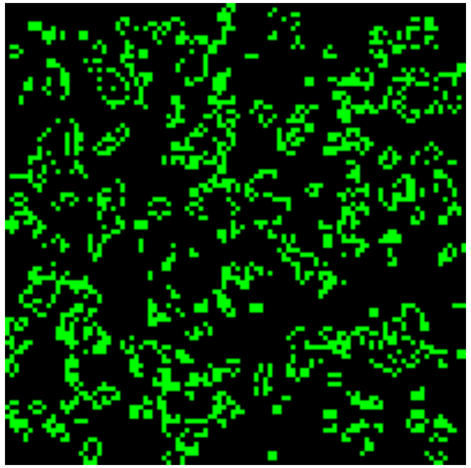
- Provides some utilities for converting data into concrete pixel values

```
// enable unqualified access to the constants in enum ‘imageType’
use imageType;
// read image as a 2D Chapel array
var img = readImage(myImageFile, jpg);
// process the image
// …
// write a scaled-up version out to a file
writeImage("outImage.png", png, scale(img, 2));
```

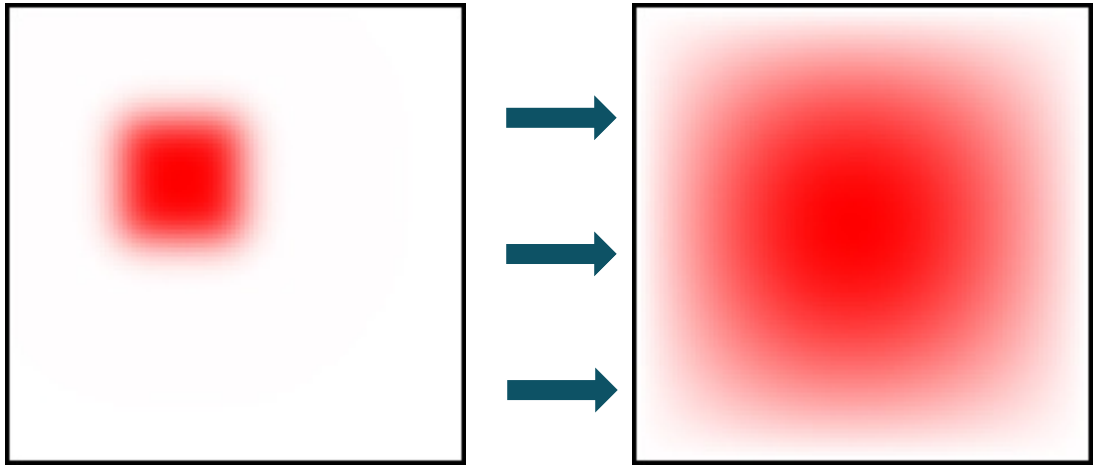
‘Image’ Module

Impact and Next Steps

Impact: Able to create basic visualizations in native Chapel



Conway’s Game of Life



2D Discrete Heat Diffusion

Next Steps:

- Provide higher-level abstractions for image manipulation
  - Drawing shapes on a ‘canvas’
  - Plotting data

Other Language/Library Improvements

Other Language/Library Improvements

For a more complete list of language and library changes and improvements in the 2.1 and 2.2 releases, refer to the following sections in the CHANGES.md file:

- New Language Features
- Language Feature Improvements
- Semantic Changes / Changes to the Language Definition
- Syntactic / Naming Changes
- Deprecated / Unstable / Removed Language Features
- New Standard Library Features
- New Package Module Features
- Changes / Feature Improvements in Standard Libraries
- Changes / Feature Improvements in Package Modules
- Standard Layouts and Distributions
- Name Changes in Libraries
- Deprecated / Unstable / Removed Library Features
- Documentation Improvements
- Language Specification Improvements
- Technical Note Improvements
- Documentation Improvements for Libraries
- Error Messages / Semantic Checks
- Bug Fixes
- Bug Fixes for Libraries
- Developer-oriented changes: Documentation
- Developer-oriented changes: Module changes

Thank you

https://chapel-lang.org
@ChapelLanguage