Library Additions and Improvements

Chapel Team, Cray Inc.

Chapel version 1.18

September 20, 2018

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

- **New Modules**
  - HDF5 Package Module
  - NetCDF Package Module
- **Module Improvements**
  - LinearAlgebra Package Module
  - Crypto Package Module
  - Error-Handling in Standard Modules

New Modules

HDF5 Package Module

HDF5: Background

- **HDF5 is a popular data storage format**
  - "Flexible and efficient I/O and high volume, complex data"
  - Widely used in scientific codes
- **Library is written in C**
- **Chapel should be able to read and write HDF5 files**

HDF5: This Effort

- **Provide an interface for reading/writing HDF5 files**
  - Call HDF5 functions using 'extern's in the 'C_HDF5' sub-module

```
use HDF5.C_HDF5;
// Open the file, read the 64-bit integers into array 'data'
var data: [0..#numElements] int;
const fid = H5open("data.h5", H5F_ACC_RDONLY, H5P_DEFAULT);
H5LTread_dataset(fid, c"dset", H5T_STD_I64LE, c_ptrTo(data));
H5Fclose(fid);
```

- **Provide higher-level functions for reading/writing files**
  - Higher-level functions are in the main 'HDF5' module
    - Read data in chunks
    - Support distributed arrays
    - Apply user-defined preprocessing
    - In parallel

HDF5:

- Example – Reading Block and Cyclic distributed arrays

```
use HDF5, BlockDist, CyclicDist;

const Space = {1..100};
const BlkSpace = Space dmapped Block(boundingBox=Space),
      CycSpace = Space dmapped Cyclic(startIdx=Space.low);

var BlkArr: [BlkSpace] int, CycArr: [CycSpace] int;

hdf5ReadDistributedArray(BlkArr, fileName, blkDsetName);
hdf5ReadDistributedArray(CycArr, fileName, cycDsetName);
```

Regular distributed domain/array declarations

Single function to read arrays for multiple distributions

HDF5: Status and Next Steps

Status:

- HDF5 library is available to call from Chapel
- Several routines written in Chapel to extend the functionality
  - Parallel reads/writes
  - Chunked reads
  - Reads to distributed arrays
  - Preprocessing capabilities

Next Steps:

- Continue to extend the functionality in Chapel
  - Support writes in the same ways as reads
  - Support more distributions
  - Support more file access patterns

NetCDF Package Module

NetCDF

Background: NetCDF is a popular data storage format

- For "creation, access, and sharing of array-oriented scientific data"

This Effort: Provide an interface for reading/writing NetCDF files

- Create 'extern' functions to enable calling the C library from Chapel

```
config const filename = "data.nc";
var ncid, varid: c_int, data: [1..NX, 1..NY] c_int;
nc_open(filename.c_str(), NC_NOWRITE, ncid); // open file for reading
nc_inq_varid(ncid, c"data", varid);          // get the ID for "data"
nc_get_var_int(ncid, varid, data[1,1]);      // read values for "data"
nc_close(ncid);                              // close the file
```

Impact: Chapel programs can now read and write NetCDF files

Next Steps: Create functions for more convenient access to data

- e.g., parallel/distributed reads/writes, data preprocessing capabilities

Module Improvements

LinearAlgebra Package Module

LinearAlgebra: Background

- **Provides a high-level interface for linear algebra**
  - Design influenced by NumPy and MatLab
  - Provides helper functions for creating matrices and vectors as arrays
  - Supports many linear algebra operations on matrices and vectors
- **Implementations use both Chapel and external libraries**
  - Some examples:
    - Dot product (Chapel)
    - Matrix-matrix multiplication (BLAS)
    - Cholesky decomposition (LAPACK)
- **'Sparse' submodule supports sparse linear algebra**
  - Supports a subset of the features available in Linear Algebra module

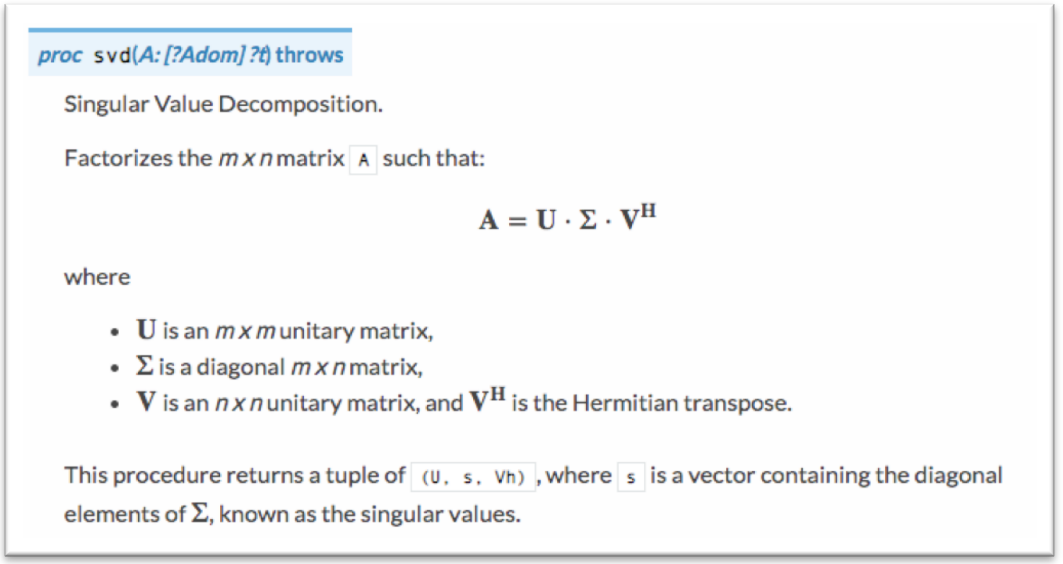
LinearAlgebra: This Effort

- **Improved feature set**
  - Added svd() procedure (singular value decomposition)
- **Improved usability**
  - BLAS and LAPACK dependencies only required if actually used
  - New BLAS and LAPACK flags allow for disabling dependencies
  - Shape preservation for promoted operations
- **Improved performance**
  - Optimized sparse matrix-matrix multiplication
  - Improved sparse matrix-matrix addition
    - Contributed by Kerim Tshimanga

LinearAlgebra: Impact - Features

- **Singular value decomposition now available**
  - Important procedure for many computations
    - e.g. PCA (principle component analysis)
  - Utilizes LAPACK

```
use LinearAlgebra;

var A = Matrix([3, 2, 2],
               [2, 3, -2],
               eltType=real);
var (U, s, Vh) = svd(A);
```



proc svd(*A: [?Adom] ?t*) throws

Singular Value Decomposition.

Factorizes the *m x n* matrix `A` such that:

$$\mathbf{A} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^{\mathbf{H}}$$

where

- $\mathbf{U}$ is an *m x m* unitary matrix,
- Σ is a diagonal *m x n* matrix,
- $\mathbf{V}$ is an *n x n* unitary matrix, and $\mathbf{V}^{\mathbf{H}}$ is the Hermitian transpose.

This procedure returns a tuple of `(U, s, Vh)`, where `s` is a vector containing the diagonal elements of Σ, known as the singular values.

LinearAlgebra: Impact - Usability

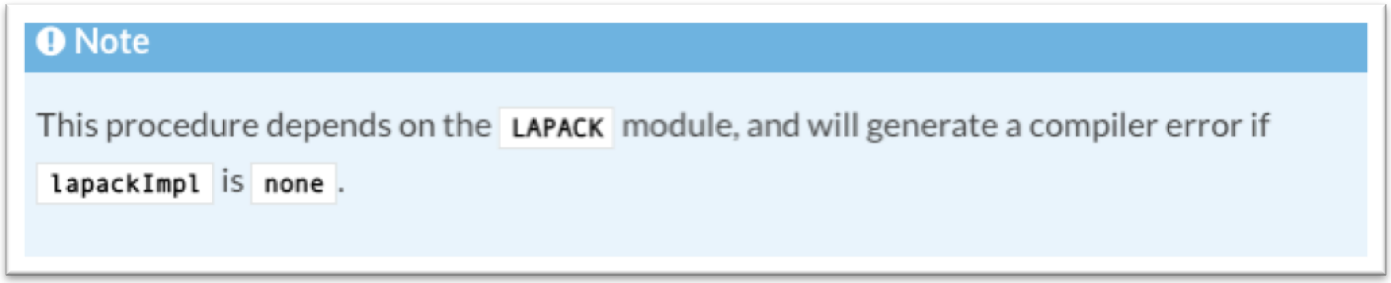
- **New config params in BLAS / LAPACK module:**
  - *blasImpl & lapackImpl*
    - Allows users to easily toggle BLAS/LAPACK implementations used
    - Sets header for common implementations (none, blas, mkl)

      ```
      chpl -s blasImpl=mkl -s lapackImpl=mkl
      ```

  - *blasHeader & lapackHeader*
    - Allows users to try previously untested BLAS/LAPACK implementations
    - Sets header explicitly for any implementation, overrides **Impl* flags

      ```
      chpl -s blasHeader=\"gsl_blas.h\"
      ```

- **Dependencies documented:**



LinearAlgebra: Impact - Usability

- **Dependencies only required when they are used:**

```
use LinearAlgebra;

// No usage of BLAS or LAPACK
var A = Matrix([0.0, 1.0, 1.0],
               [1.0, 0.0, 1.0],
               [1.0, 1.0, 0.0]);
var I = eye(3,3);
var B = A + I;
var A = Matrix(10, 4);

// 1.17: chpl –lblas –llapack example.chpl
// 1.18: chpl example.chpl
```

LinearAlgebra: Impact - Usability

- **Dependencies only required when they are used:**

```
use LinearAlgebra;

// eigvals() requires LAPACK
var A = Matrix([2.0, 1.0], [1.0, 2.0]);
var (eigenvalues, eigenvectors) = eigvals(A, right=true);
```

```
// 1.17: chpl –lblas –llapack example.chpl
// 1.18: chpl –llapack example.chpl
```

LinearAlgebra: Impact - Usability

- **Dependencies only required when they are used:**

```
use LinearAlgebra;

// mat-mat mult requires BLAS unless `–s BlasImpl=none` is set
var A = Matrix(3,5);
A = 2;
var AA = A.dot(A.T);
```

```
// 1.17 (BLAS):    chpl –lblas –llapack example.chpl
// 1.18 (BLAS):    chpl –lblas example.chpl
// 1.18 (native):    chpl –s blasImpl=none example.chpl
```

LinearAlgebra: Impact - Usability

- **Promoted operations now preserve shape**
  - Removed caveats about this in documentation
  - Element-wise operation methods not needed for dense matrices
    - Allows more natural usage of operations: +, -, *, /

```
// 1.17 element-wise operations
use LinearAlgebra;

var A = Matrix(3,3),
    B = Matrix(3,3);

var C = A.plus(B);
```

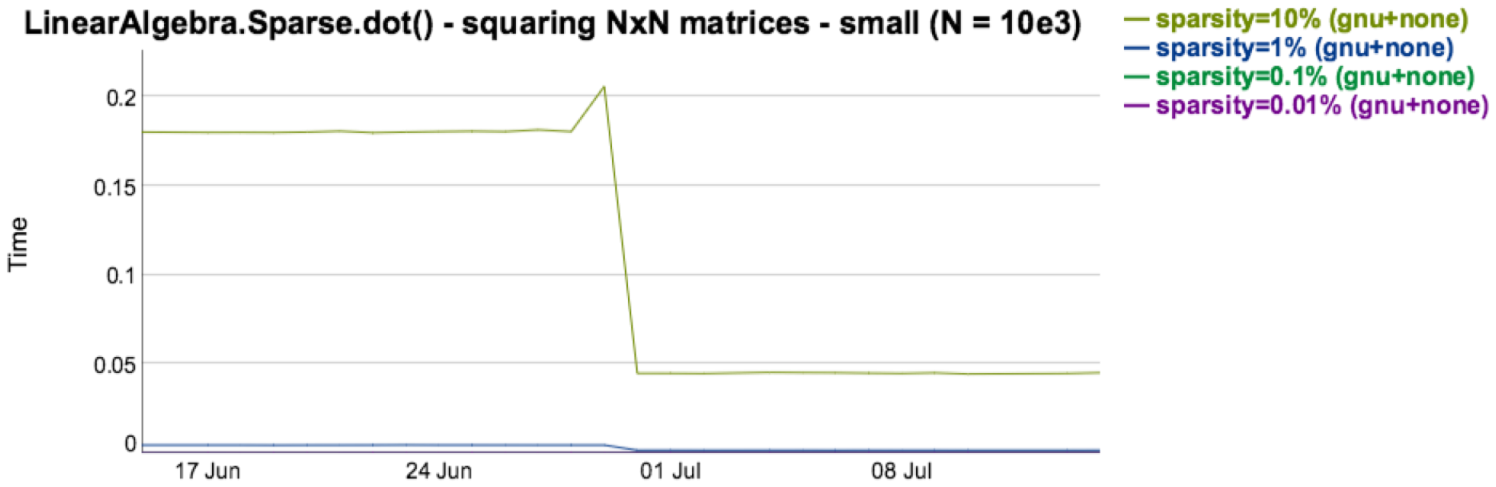
```
// 1.18 element-wise operations
use LinearAlgebra;

var A = Matrix(3,3),
    B = Matrix(3,3);

var C = A + B;
```

LinearAlgebra: Impact - Performance

- **Sparse matrix-matrix multiplication**
  - Switched sparse matrices to default to unsorted indices

    // Default array layout in LinearAlgebra.Sparse

    ```
    CS(compressedRows=true, sortedIndices=false)
    ```
  - Performance improvement scales with matrix density



LinearAlgebra: Next Steps

- **Distributed support**
  - Considering interfacing with ScaLAPACK
- **GPU support**
  - Considering interfacing cuBLAS or clBLAS
- **More features for dense and sparse modules**
- **More next steps tracked in issue #5753**

Crypto Package Module

Crypto

Background: Crypto package module uses OpenSSL

- Implemented with OpenSSL 1.0
- OpenSSL 1.1 has API changes and is in Ubuntu 18.04

This Effort: Support both OpenSSL 1.0 and 1.1 in Crypto

- Contributed by Sarthak Munshi

Impact: Crypto module more portable

Error-Handling in Standard Modules

Error-Handling: Background

- **Error handling recommended for use after 1.17**
  - However, we did not have much code that used error handling
    - made it difficult to build confidence in design and implementation
- **Want to avoid halts in standard modules**
  - Halts are hostile to users and prevent them from addressing errors
  - Halting acceptable only…
    …for unrecoverable problems
    …when performance penalty of error handling is too high
  - In particular, halting is currently considered acceptable for things like:
    - out-of-memory
    - bounds checking
    - cast checking
    - nil checking

Error-Handling: This Effort

- **Removed halts from remaining standard modules**
  - Converted “out error” pattern to error handling
  - Converted halts in several modules to error handling
    - DateTime, BigInteger, IO, Barrier, and a few others
  - Improved implementation in cases where halts were unnecessary
  - Downgraded some halts to warnings
    - calling start() on an already started timer
    - using `numTasks < 0` for dynamic iterators
- **Converted Mason to use error-handling**
- **Started on initial “users guide” for error-handling**
  - When to use error-handling and performance considerations
    - See #10703

Error-Handling: Impact and Next Steps

Impact:

- Improved quality of the standard modules
- Gained a bit more confidence in error-handling design/implementation
  - particularly from converting mason

Next Steps:

- Continue to put more weight on error-handling
- Remove halts from internal modules and distributions
- Formalize error-handling users guide

For More Information

For a more complete list of library changes in the 1.18 release, refer to the ‘Standard Modules / Library’, ‘Package Modules’ and ‘Bug Fixes’ sections in the **CHANGES.md** file.

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