Standard Library Improvements

Chapel Team, Cray Inc.
Chapel version 1.14
October 6, 2016

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**
  - BigInteger
  - RangeChunk
  - BLAS
  - MPI
  - ZeroMQ
  - MatrixMarket
- **Module Improvements**
  - Sort/Search: Interface and Functionality Changes
  - Math: Standard Functions for Complex Types
  - String: New join() and split() Methods
  - DynamicIters: Dynamic Domain Iterators
  - Reflection: Extended Query Functionality
  - Other Library Improvements

New Modules

BigInteger

BigInteger: Background

- **Chapel provides a GMP (GNU Multi Precision) module**
  - Implemented by wrapping C GMP library
  - Provides low-level access to multi-precision math:
    - Signed integers (mpz) with ~140 arithmetic/logical functions
    - Floating-point (mpf) with 15, out of ~70, functions
    - Random numbers (gmp_random_state) with 15 functions
  - Many GMP functions rely on side-effects
    - One or more actuals are modified and return type is void
    - Common to define variations with scalar arguments
    - Aims to reduce number of init/free operations

BigInteger: Background

- **GMP.chpl has also included a prototype 'BigInt' class**
  - Approximately ~115 methods that wrap equivalent GMP functions
  - Drawbacks:
    - User responsible for freeing class instances
    - Limited support for arithmetic expressions
      - Methods tend to have void return type

BigInteger: This Effort

- **Implement record `bigint'**
  - Wraps mpz
  - Manages memory for GMP state
  - Includes methods that are thin wrappers over GMP functions
- **Define operator overloads for arithmetic/logical operators**
  - Symmetric support for scalar and bigint actuals

```
var x = new bigint(2);
var a, b, c : bigint;

a = x * x;
b = x + 4;
c = 9 + x;
```

  - Compound assignment operators comparable to GMP primitives

```
proc +=(ref a: bigint, const ref b: bigint) : void ...
```

BigInteger: Impact

- **An inner loop of pidigits with class BigInt**

```
do {
  k += 1;
  mpz_addmul_ui(accum, numer, 2);        // accum += numer * 2
  mpz_mul_ui(numer, numer, k);           // numer *= k
  mpz_mul_ui(denom, denom, 2 * k + 1); // denom *= (2k+1)
  mpz_mul_ui(accum, accum, 2 * k + 1); // accum *= (2k+1)
} while (mpz_cmp(numer, accum) > 0);   // while numer > accum
```

- **‘bigint’ version performs comparably with better clarity**

```
do {
  k     += 1;
  accum += numer * 2;
  numer *= k;
  denom *= (2 * k + 1);
  accum *= (2 * k + 1);
} while (numer > accum);
```

BigInteger: Status and Next Steps

Status:

- class BigInt is deprecated for 1.14
  - warning emitted during compilation if a BigInt is constructed
  - will be dropped in 1.15
- record bigint is preferred API for 1.14
  - freedom from user-defined memory management
  - improves expressiveness with broad set of operator overloads

Next Steps:

- Reduce overhead caused by operators that introduce temps
  - primarily due to malloc/free and deep copies on assignment
  - related to optimizations for procedures returning local arrays
    - (see ongoing efforts deck)

Range Chunking

Range Chunking: Background

- **No simple interface to divide a range into *n* chunks**
  - Requires falling back on more manual approaches:

```
coforall chunk in 1..numChunks {
  // _computeChunkStartEnd is buried in DSIUtil, does not take ranges, returns only indices
  const (startIx, endIx) = _computeChunkStartEnd(
                                          numElems, numChunks, chunk);
  for i in startIx..endIx do
    yield indices(i);
}
```

  - Users requested standard library support for such cases
  - Distributions currently implement this logic *ad hoc* / inconsistently

Range Chunking: This Effort

- **Offer a richer public interface for range chunking**
  - Take in ranges as input arguments
  - Support both query and iterator interfaces
  - Return chunks as ranges or indices

```
use RangeChunk;
```

- **Make use of the library in our standard distributions**
  - Block-cyclic
  - Replicated
  - Default sparse
  - CSR layout

Range Chunking: Impact

- **More elegant, legible code**
  - Here is the example from earlier:

```
coforall chunk in 1..numChunks {
  const (startIx, endIx) = _computeChunkStartEnd(
                              numElems, numChunks, chunk);
  for i in startIx..endIx do
    yield indices(i);
}
```

  - Here is the example rewritten with the new library:

```
coforall chunk in chunks(1..numElems, numChunks) {
  for i in chunk do
    yield indices(i);
}
```

Range Chunking: before the library

```
// CSR layout range that needs chunking
const hereDenseInds = 0:resultIdxType..#wholeR.length
                                        by nLocs align AL;

// Manual adjustment for stride, index type, taskid/chunks
const hereNumInds = hereDenseInds.length;
const hereFirstInd = hereDenseInds.first;
const (begNo, endNo) = _computeChunkStartEnd(hereNumInds, numTasks,
                                                          taskid+1);

const begIx = (hereFirstInd + (begNo - 1) * nLocs):resultIdxType;
const endIx = (hereFirstInd + (endNo - 1) * nLocs):resultIdxType;
assert(hereDenseInds.member(begIx));
assert(hereDenseInds.member(endIx));

return begIx .. endIx by nLocs;
```

Range Chunking: with the library

```
// CSR layout range that needs chunking
const hereDenseInds = 0:resultIdxType..#wholeR.length
                                 by nLocs align AL;

return chunk(hereDenseInds, numTasks, taskid);
```

Range Chunking: Status and Next Steps

Next Steps:

- Integrate the library with more distributions
  - Block is the most significant one remaining
- Receive and incorporate user feedback

BLAS

BLAS: Background

- **Linear Algebra is core to a vast number of applications**

- **Chapel provides: LAPACK and LinearAlgebraJAMA**
  - Neither of which supports common ops such as matrix-multiplication

- **Basic Linear Algebra Subprograms (BLAS)**
  - The *de facto* standard for low-level linear algebra operations
  - Commonly used in conjunction with LAPACK
  - Variety of implementations support a variety of parallelism
    - Multicore: OpenBLAS, ATLAS, MKL
    - Distributed: PBLAS, DBLAS
    - GPU: CuBLAS, clBlas

BLAS: This Effort

- **BLAS has been ported to Chapel as C_BLAS and BLAS**
  - C_BLAS
    - Low-level API, wraps all CBLAS routines (which wrap BLAS)
    - Many arguments are C types (c_int, c_string, …)

```
extern proc cblas_dgemm(…TransA: c_int, M: c_int, N: c_int, …
                        A:[] c_double, …)
```

  - BLAS
    - High-level API, wraps C_BLAS, only supports level-3 operations
    - Generic across all matrix element types: real(32|64), complex(64|128)
    - Many arguments with obvious defaults made optional
    - some defaults are taken from array meta-data (e.g., M, N, ldA)

```
proc gemm(A: [?Adom] ?t, … opA = Op.N, …)
```

- **Requires an underlying BLAS installation**
- **Contributed by Nikhil Padmanabhan**

BLAS: Status and Next Steps

Status: BLAS module available in 1.14

- BLAS level-3 routines are available through BLAS module
- Features and build instructions well-documented
- Supports many BLAS implementations

Next Steps: Extend BLAS support

- Support BLAS level 1 & 2 routines
- Improve ease of installation/use
- Deploy BLAS/LAPACK similar to HDFS as a build configuration
- Build a linear algebra module on top of BLAS & LAPACK

MPI

MPI: Background

- **MPI is the *de facto* standard message-passing library**
  - A foundational technology for the HPC community

- **Chapel has talked about adding MPI support for years**
  - to support interoperability with MPI codes
  - to provide familiar capabilities for MPI experts
  - as a way to manage communication more explicitly within Chapel

MPI: This Effort

- Provide a high level interface to MPI via MPI module
- Provide a low level interface to MPI via C_MPI submodule
- Allow launching MPI+Chapel programs in an SPMD mode
- Contributed by Nikhil Padmanabhan

MPI: This Effort

- **MPI module provides high-level MPI interface in Chapel**
  - Restricted to certain configurations (e.g., doesn't work with qthreads)
- **C_MPI submodule provides C-API for MPI 1.1 standard**
  - ± some routines
- **MPI code in Chapel can be compiled in two modes:**
  - **Multilocale Mode** (`CHPL_COMM != none`)
    - Locales are treated as MPI ranks
    - Program is launched like any other Chapel program
  - **SPMD Mode** (`CHPL_COMM == none`)
    - Multiple copies of program are run simultaneously like true MPI ranks
    - Program must be launched with mpirun launcher or manually
- **mpirun launcher modified to support flag: --spmd**
  - Allows launching with multiple SPMD ranks when `COMM=none`
  - Users explicitly specify SPMD ranks (--spmd) or locales (-nl)

MPI: Impact

- **Chapel users who know MPI have an easier starting point**

- **MPI can be used within Chapel for explicit communication**
  - low level programming / optimized communication

MPI: SPMD Example

```
use MPI;

var rank = commRank(CHPL_COMM_WORLD),
    size = commSize(CHPL_COMM_WORLD);

for irank in 0.. #size {
  if irank == rank then
    writef("MPI Hello! This is rank=%i of size=%i, on locale.id=%i\n",
           rank, size, here.id);
  C_MPI.MPI_Barrier(CHPL_COMM_WORLD);
}
```

CHPL_TARGET_COMPILER=cray-prgenv-gnu
CHPL_TASKS=fifo
CHPL_COMM=none

MPICH_MAX_THREAD_SAFETY=multiple

```
> chpl hello.chpl -o hello && ./hello --spmd=4
```

```
Hello! This is MPI rank=0 of size=4, on locale.id=0
Hello! This is MPI rank=1 of size=4, on locale.id=0
Hello! This is MPI rank=2 of size=4, on locale.id=0
Hello! This is MPI rank=3 of size=4, on locale.id=0
```

MPI: Multilocale Example

```
use MPI;

coforall loc in Locales do on loc {
  var rank = commRank(CHPL_COMM_WORLD),
      size = commSize(CHPL_COMM_WORLD);

  for irank in 0.. #size {
    if irank == rank then
      writef("Hello! This is MPI rank=%i of size=%i, on locale.id=%i\n",
                rank, size, here.id);
    C_MPI.MPI_Barrier(CHPL_COMM_WORLD);
  }
}
```

CHPL_TARGET_COMPILER=cray-prgenv-gnu
CHPL_TASKS=fifo
CHPL_COMM=gasnet
CHPL_COMM_SUBSTRATE=mpi

MPICH_MAX_THREAD_SAFETY=multiple
AMMPI_MPI_THREAD=multiple

```
> chpl hello.chpl -o hello && ./hello --nl=4
Hello! This is MPI rank=0 of size=4, on locale.id=0
Hello! This is MPI rank=1 of size=4, on locale.id=1
Hello! This is MPI rank=2 of size=4, on locale.id=2
Hello! This is MPI rank=3 of size=4, on locale.id=3
```

MPI: Status and Next Steps

Status: MPI module available in 1.14 release

- Still considered a work in progress
  - Limited configuration support
  - Tested with MPICH
- Documented in package modules

Next Steps: Improve MPI module

- Support more configurations
  - Qthreads tasking layer
  - --spmd for all launchers
  - Other MPI implementations (OpenMPI)
- Expand high-level interface
  - Currently limited to setup/cleanup routines
- Support 1-sided MPI operations / newer MPI methods

ZeroMQ

ZeroMQ: Background

- ZeroMQ is a light-weight messaging library.

http://zeromq.org/

Distributed Messaging

ZeroMQ \zero-em-queue\, \ØMQ\:

- Ø Connect your code in any language, on any platform.
- Ø Carries messages across inproc, IPC, TCP, TIPC, multicast.
- Ø Smart patterns like pub-sub, push-pull, and router-dealer.
- Ø High-speed asynchronous I/O engines, in a tiny library.
- Ø Backed by a large and active open source community.
- Ø Supports every modern language and platform.
- Ø Build any architecture: centralized, distributed, small, or large.
- Ø Free software with full commercial support.

ZeroMQ: Background & This Effort

Background:

- Chapel has good support for tightly-coupled parallel programs
  - What about other use cases?
- ZeroMQ bindings offer one way to enable:
  - Chapel programs fit into distributed work-flows
  - Chapel programs work with programs in other languages
  - Chapel programs can be servers
  - Chapel programs can scale up and down depending on their input

This Effort: Implement high-level bindings to ZeroMQ

- Contributed by Nick Park

ZeroMQ: Impact

- Basic functionality is available
- Example shows communication through ZeroMQ sockets

```
// pusher.chpl
use ZMQ;
config const to: string = "world!";
var context: Context;
var socket = context.socket(ZMQ.PUSH);
socket.bind("tcp://*:5555");
socket.send(to);
```

```
// puller.chpl
use ZMQ;
var context: Context;
var socket = context.socket(ZMQ.PULL);
socket.connect("tcp://localhost:5555");
writeln("Hello, ", socket.recv(string));
```

ZeroMQ: Status and Next Steps

Status:

- ZeroMQ Context and Sockets are available
- Socket supports send and receive
- Send and receive block only the calling tasks
  - other tasks can use that core while that task waits
- Works with multilocale Chapel programs

- not implemented yet:
  - ZeroMQ message objects
  - handling errors
  - explicitly non-blocking send/recv calls

Next Steps:

- complete implementation
- provide full ZeroMQ functionality

MatrixMarket

MatrixMarket

Background: MatrixMarket format

- MatrixMarket website http://math.nist.gov/MatrixMarket/
- includes a library of test sparse matrices
- defines text formats for sparse and dense matrices

This Effort: Support MatrixMarket format

```
// read a sparse matrix
var A = mmreadsp(complex, filename);
// write a sparse matrix
mmwrite("test.mtx", A);
```

- Contributed by Chris Taylor

Impact: Better interoperability with Python and Matlab

Next Steps: Improve documentation, include in built docs

Module Improvements

Sort/Search

Sort/Search: Background and This Effort

Background: Sort and Search modules works-in-progress

- Unnecessary routine arguments and data requirements
- No comparator support
- Developed as proof-of-concepts; not optimized for performance

This Effort: Improve Sort and Search modules

- Broaden functionality
- Improve interface
- Track performance

Sort/Search: Impact

- **Comparator argument added to Search/Sort interface**
  - Comparators are instances of class/records that have a method:
    - Comparator.key(a) - returns any type that supports < operator
    - Comparator.compare(a, b) - returns signed type
- **Interface simpler and more consistent**
  - Dropped ‘doublecheck: bool’
  - Comparator replaces ’reverse: bool’
  - Switched to camelCase naming
    - QuickSort() -> quickSort(), LinearSearch -> linearSearch()
  - VerifySort renamed isSorted()
- **More helpful error messages for incorrect usage**
- **Performance testing for Sort routines**
- **Progress towards strided & non-aligned array support**

Sort/Search: Next Steps

- **Improve parallelism of implementations**
  - Provide a well defined interface for controlling degree of parallelism
  - Tune implementations for distributed arrays
- **Add new implementations**
  - radixSort
  - timSort (hybrid merge/insertion sort)
- **Broaden functionality**
  - Non-aligned strided arrays
  - Multi-dimensional arrays
- **Consider adding FCF comparator support**
  - Improves simplicity and conciseness
- **Promote Sort / Search to "Standard Modules"**

Standard Math Functions for Complex Types

Standard Math Functions for Complex Types

Background: Many math routines missing for complex numbers

This Effort: Added C99 function wrappers on complex numbers

Complex trigonometric and hyperbolic functions

```
sin(z),   cos(z),   tan(z)    // trig functions
asin(z),  acos(z),  atan(z)   // arc trig functions
sinh(z),  cosh(z),  tanh(z)   // hyperbolic functions
asinh(z), acosh(z), atanh(z)  // arc hyperbolic functions
```

Complex manipulation, exponential, power functions

```
abs(z)    // absolute value
carg(z)   // phase angle
conjg(z)  // conjugate
cproj(z)  // projection onto Riemann sphere
exp(z)    // base-e exponent
log(z)    // natural logarithm
sqrt(z)   // square root
```

Impact: All of the functions above callable on complex numbers

String join(), split()

String join(), split()

Background: These string methods were not intuitive

- `join()` only accepted an array of strings
- `split()` could only delimit by a single space

This Effort: Make these methods more flexible

- `join()` accepts varargs, homogenous tuples, and arrays of strings
- `split()` can delimit by arbitrary amounts of whitespace

Impact: Strings are easier to manipulate

Dynamic Domain Iterators

Dynamic Domain Iterators

Background:

- ‘DynamicIters’ module supports a dynamic() iterator over ranges
  - similar to ‘dynamic’ scheduling in OpenMP

This Effort:

- extend dynamic() iterator to support domains as well
  - user can select dimension to parallelize; defaults to the first

Impact:

- avoids the need to manually decompose multidimensional loops:

  e.g., rather than:

```
forall x in dynamic(D.dim(1), chunkSize) do
  for y in D.dim(2) do
```

  …can now write:

```
forall (x,y) in dynamic(D, chunkSize) do
```

- made use of this in the Mandelbrot shootout benchmark

Dynamic Domain Iterators

Next Steps:

- extend dynamic() to support other types as well? (e.g., arrays)
- extend other 'DynamicIters' iterators to support domains
- streamline the effort required to apply a range iterator to a domain?
  - e.g., would first-class iterators reduce the amount of coding effort required?

Reflection

Reflection Improvements

Background:

- version 1.13 introduced a ‘Reflection’ module supporting:
  - the ability to reason about object fields (e.g., query types, get values)
  - the ability to query whether a given function call could be made / resolved

This Effort:

- extended object field queries to support getting ‘ref’s to fields

```
class C { var x, y: int; }
var myC = new C(x=2, y=3);
getFieldRef(myC, 1) = 4;      // assign 4 to myC.x via field number
getFieldRef(myC, "y”) = 5;    // assign 5 to myC.y via field name
```

- extended function resolution queries to include type methods

```
proc type C.foo(x: int) { … }
if canResolveTypeMethod(C, "foo”, 3) then …
```

Reflection Improvements

- **Impact:**
  - expands user ability to reflect about Chapel code at compile-time
  - made use of new features in Chapel's internal modules

- **Next Steps:**
  - continue to make improvements to 'Reflection' based on experience

Other Library Improvements

Other Library Improvements

- Changed *ascii()* return types from *int(32)/int(64)* to *uint(8)*
- New Buffer.copy{in,out} support for strings
  - Contributed by Nick Park
- Removed library functions on *c_strings*
- Changed default binary string format to *data_toeof*
- Added bounds-checking to string index/slice ops
- Added bounds-checking to array vector operations

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 CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY