Library Improvements

Chapel Team, Cray Inc.

Chapel version 1.16

October 5, 2017

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**
  - Cryptography
  - TOML
  - Parallel Collections
  - Distributed Dynamic Iterators
- **Module Improvements**
  - ZMQ
  - LinearAlgebra
  - MPI Interoperability
  - C Interoperability Improvements
  - Other Library Improvements

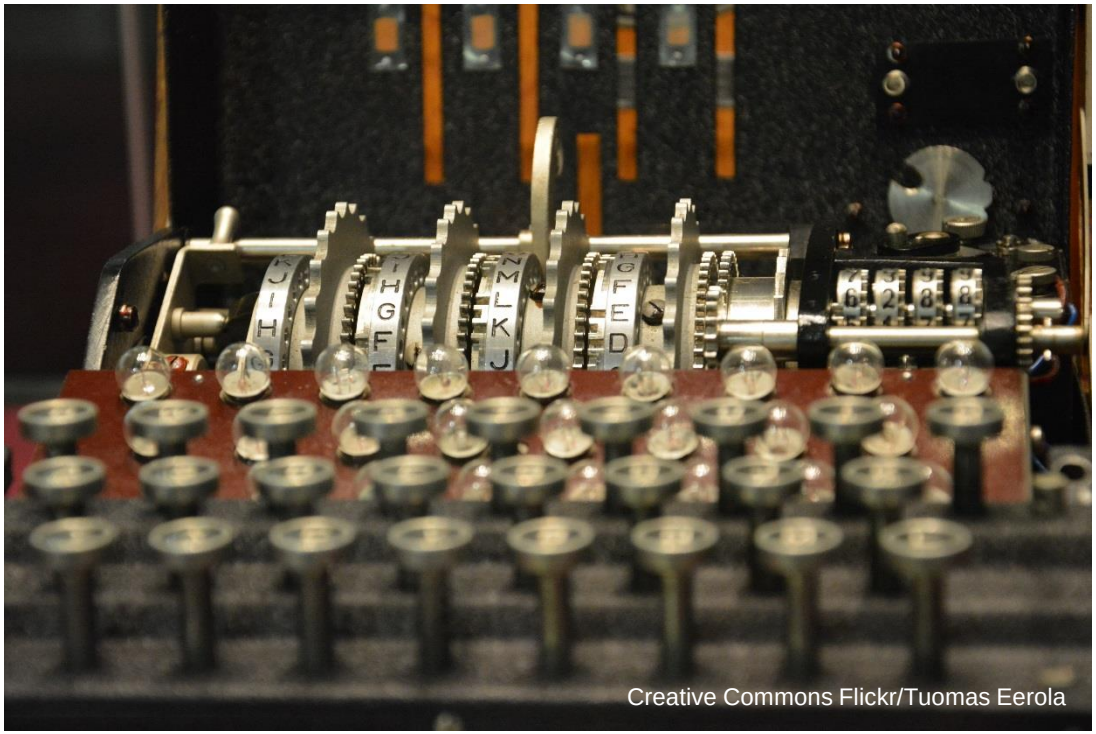
New Modules

Cryptography

Contributed by Sarthak Munshi as a GSoC project

Cryptography: Background

- Chapel had no built-in support for cryptography
- Desirable to natively encrypt/decrypt/hash



Cryptography: This Effort

- **Implemented a new Cryptography module**
  - Built on top of a new wrapper for the OpenSSL library
  - Includes several cryptography tools
    - Symmetric Ciphers (AES)
    - Asymmetric Ciphers (RSA)
    - Hashing functions (MD5, SHA, RIPEMD)
    - Key Derivation Functions (PBKDF2)
    - Cryptographically secure random number generator (CryptoRandom)

- **Google Summer of Code Project**

Cryptography: Impact

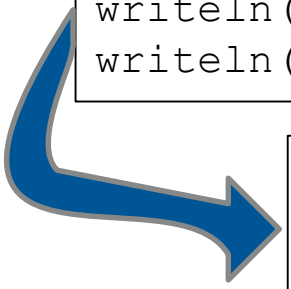
- **Much OpenSSL functionality is now available in Chapel**

```
use Crypto;
config const message = "secret message";

const aes = new AES(256, "cbc"),
      msg = new CryptoBuffer(message);
      // Also define salt, IV, hash, key

const ct = aes.encrypt(msg, key, IV);
      plaintext = aes.decrypt(ct, key, IV);

writeln("original: ", toString(msg));
writeln("encrypted: ", toString(ct));
writeln("decrypted: ", toString(plaintext));
```



```
original: secret message
encrypted: ��$%��U┐#E^~�
decrypted: secret message
```

Cryptography: Status & Next Steps

Status: Cryptography module is now available

- Routines to encrypt and decrypt
- Secure hash functions
- Secure pseudo-random number generation

Next Steps: Add extra functionality from OpenSSL

- Additional cipher algorithms
  - ECC, DES, Blowfish, Twofish
- Additional key derivation functions
- Consider switching from classes to records / ’Owned’ classes
  - goal: avoid need for ’delete’

TOML

TOML

Background: TOML is a popular markup language

- "Tom's Obvious, Minimal Language"
- TOML was chosen as the language for mason's manifest & lock files
  - So Chapel needed a TOML reader/writer module

This Effort: Created a TOML module

- Example Usage:

```
use TOML;

var TomlFile = open("Mason.toml", iomode.r);
// Parses TOML file into Toml data structure – also accepts channel or string
var TomlData = parseToml(TomlFile);

var version = TomlData["version"].toString(); // Reads a value
TomlData["version"] = "2.0.1"; // Writes a value
writeln(TomlData); // Writes to stdout in TOML format
delete TomlData; // Clean up
```

TOML

Status: Chapel has a TOML module

- Mason uses this module to read and write manifest & lock files
- Majority of TOML spec implemented
- Undocumented due to a desire to modify interface

Next Steps: Complete TOML module implementation

- Finish implementing TOML specification
  - Notably, arrays of tables
- Finalize interface design and add to public documentation
- Avoid explicit memory management through use of Owned module

Parallel Collections

Contributed by Louis Jenkins as a GSoC project

Parallel Collections

Background: A goal is to support any parallel algorithm

- And in particular to support global view programming
- One common global-view idiom: a work queue
  - for distributing work among existing tasks
- Chapel supports work queues for tasks themselves, but
  - those work queues are local only
  - the work queues can only contain tasks, not other work items

Parallel Collections: This Effort

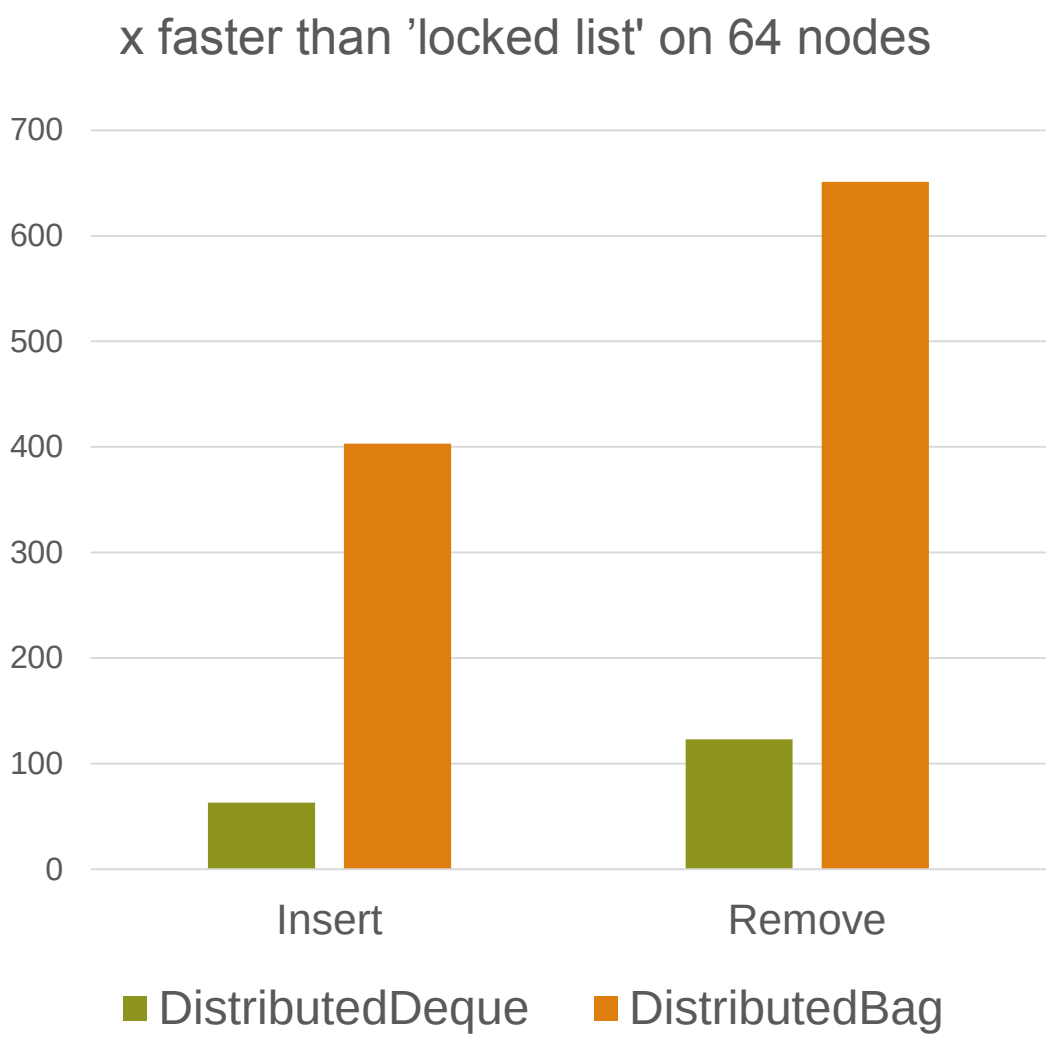
- **3 new package modules support the work queue idiom:**
  - Collections – describes interface; asserts:
    - Data structure is parallel-safe
    - Data structure supports insertion, removal, and iteration
  - DistributedBag – "work queue" with relaxed ordering

```
var c = new DistBag(int, targetLocales=Locales);
for i in 1..10 do c.add(i); // order not preserved
var counter: atomic int;
forall elem in c do counter.add(elem);
```

  - DistributedDeque – parallel FIFO/LIFO queue

```
var c = new DistDeque(int, targetLocales=Locales);
for i in 1..10 do c.add(i); // order preserved
var counter: atomic int;
forall elem in c do counter.add(elem);
```

Parallel Collections: Performance




```
coforall loc in Locales do
 on loc do
  coforall t in 0..#nTasks do
    for i in 1..n do
     c.add(i);
```

```
coforall loc in Locales do
 on loc do
  coforall t in 0..#nTasks do
    for i in 1..n do
     c.remove();
```

Parallel Collections

Impact: Significantly more performant work queue available

- Especially with multiple locales
- Even on 1 locale, DistributedBag is faster than locked list

Next Steps:

- Use in a real application
- Improve documentation
- Continue related effort of supporting distributed atomic class instances

Distributed Dynamic Iterators

Dist. Dynamic Iterators: Background

- **DynamicIters module provides OpenMP-style scheduling**
  - Dynamic, guided, and adaptive iterators
  - These work with both ranges and domains

- **No distributed load-balancing iterators available**
  - Users had to resort to writing their own iterators

Dist. Dynamic Iterators: This Effort

- **Created distributed iterators based on dynamic, guided**
  - Implemented in 'DistributedIters' package module
  - Can also be zipped with things that can follow ranges and domains
  - Can iterate over ranges and domains

```
use DistributedIters;

forall i in distributedDynamic(1..n) do
  imbalancedWorkload(i);
```

- **Users can optionally specify worker locales**
  - Default is all locales

```
const halfLocales = Locales[1..numLocales/2];
forall i in distributedGuided(1..n,
                              workerLocales=halfLocales)do
  work(i);
```

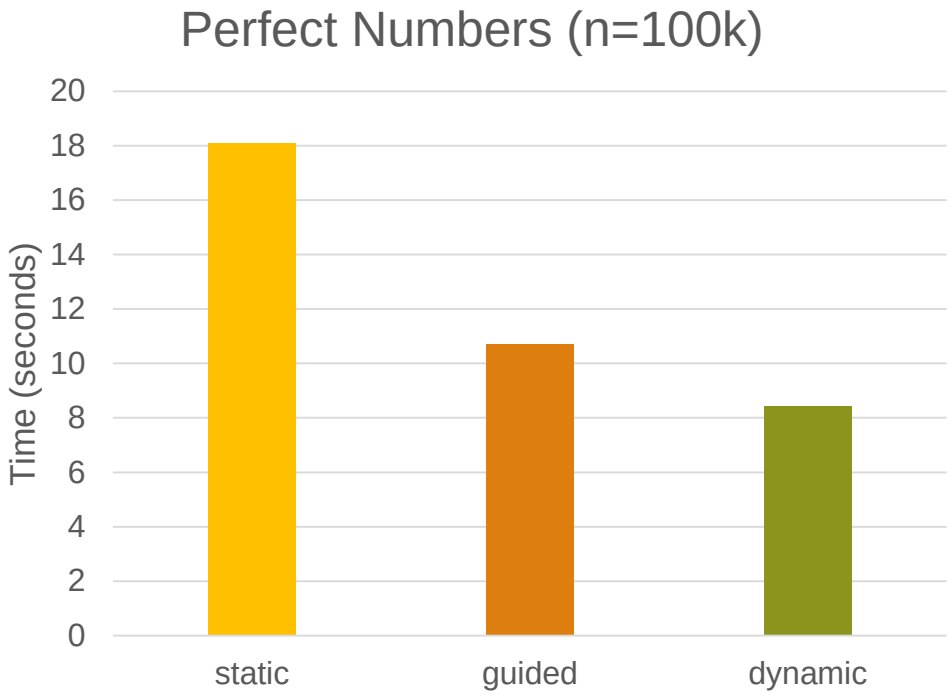
Dist. Dynamic Iterators: This Effort

- **Optionally enable/disable coordination mode**
  - 'coordinated: bool' - if true, first worker only distributes work
    - Locking/atomics often refer back to first worker
    - Can improve performance when network atomics are unavailable

- **Optionally provide chunk sizes**
  - Controls how much work each task/locale receives
  - Still looking for good defaults
  - See documentation for argument names
    - http://chapel.cray.com/docs/1.16/modules/packages/DistributedIters.html

Dist. Dynamic Iterators: Impact

- **Options now exist for distributed load-balancing**
  - e.g., find perfect numbers in uniform random distribution from 1..n
    - Implemented as naive O(n) algorithm



    - ugni-qthreads on 16 nodes of Cray XC30

Dist. Dynamic Iterators: Status and Next Steps

Status:

- ‘DistributedIters’ included as package module in 1.16

Next Steps:

- Performance tuning
  - Finding good default chunk sizes
  - Study in real-world workloads
- Support arrays in addition to domains and ranges

Module Improvements

ZMQ

ZMQ

Background: ZMQ was not 100% cross-language compatible

- Serialization of records was not compatible with REQ/REP socket
  - This caused incompatibility with other language bindings (e.g. PyZMQ)

This Effort: Reimplemented send/recv for records in ZMQ

- Confirmed this works for sending strings to PyZMQ with new tests
- Contributed by Nicholas S. Park

Impact: ZMQ enables cross-language communication

Status: ZMQ module is compatible with PyZMQ

- Functionality tested nightly

LinearAlgebra

LinearAlgebra

Background: LinearAlgebra added in 1.15

- Supported dense linear algebra operations and helper functions
- Did not support sparse linear algebra
  - Sparse linear algebra has many important applications, like graph analytics
  - User-requested feature
- Few introductory examples
  - Common request from users

This Effort: Improved LinearAlgebra module

- Added ‘Sparse’ submodule with documentation
- Added a LinearAlgebra primer
  - Includes sparse examples
- Other minor improvements

LinearAlgebra: Sparse submodule

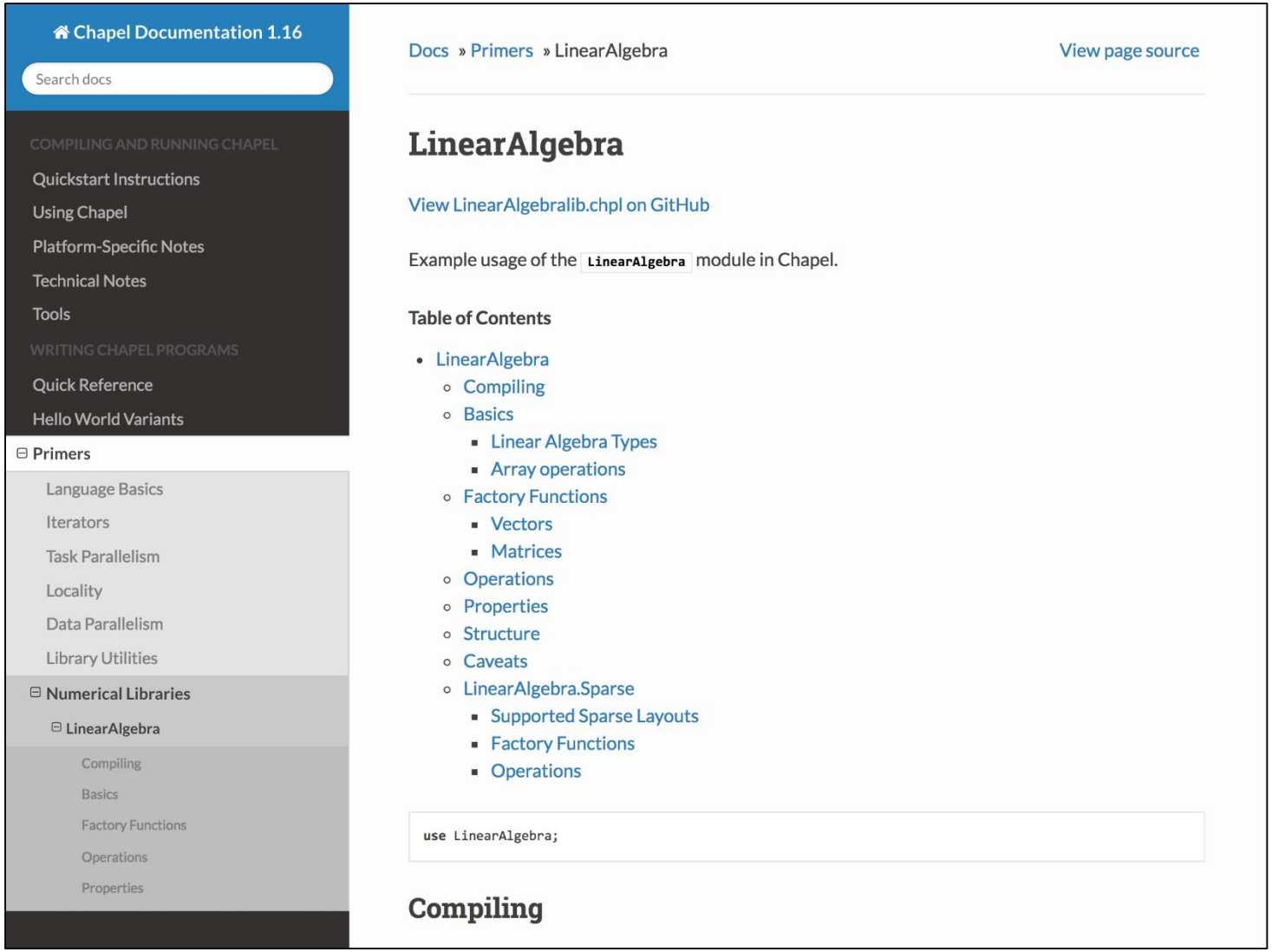
- **Subset of linear algebra features for dense matrices**
  - Uses the same interface and naming schemes

```
use LayoutCS;
use LinearAlgebra.Sparse;

var D = CSRDomain(5, 5); // empty 5x5 sparse domain
var A = CSRMatrix(D); // sparse array over 'D'
D += [(1,1), (2,2), (4,3), (3,4)]; // Add indices to domain
A = 4.0; // Set all nonzeroes

var B = A.dot(A);
for i in B.domain do writeln(i, ": ", B[i]);
// (1, 1): 16.0
// (2, 2): 16.0
// (4, 3): 16.0
// (3, 4): 16.0
```

LinearAlgebra: Added Primer



LinearAlgebra: Other Improvements

- **Documentation corrected for triangular functions**
- **diag() function contributed by Prabhanjan Mannari**
  - Extracts diagonal from matrices into a vector
  - Builds a diagonal matrix from a vector
- **Adopted array.op(arg) interface for all matrix operations:**
  - A.plus(B)
  - A.minus(B)
  - A.times(B)
  - A.elementDiv(B)
  - A.dot(B)
- **Deprecated matPlus and matMinus**
  - array.op(arg) style preferred

LinearAlgebra

Status: Sparse linear algebra and primer added in 1.16

Next Steps: Additional Linear Algebra features

- Distributed Linear Algebra
  - Dense
  - Sparse
- LAPACK support
  - Eigensolvers, SVDs, etc.
- Linear Algebra on GPUs
  - CuBLAS, clBLAS
- Improve compilation process
  - Add build option for Chapel to download BLAS and LAPACK

MPI Interoperability

MPI Interoperability: Background and Effort

Background: Could not use Qthreads with MPI

- Non-preemptive tasks could cause deadlock

This Effort: Permit using MPI with Qthreads

- Implemented blocking operations with non-blocking op + yielding-wait

```
proc Send(…){
  MPI_Isend(…);
  MPI_Test(flag, …);
  while flag == 0 {
    chpl_task_yield();
    MPI_Test(flag, …);
  }
}
```

Impact: Can now use MPI and Qthreads

- Improved performance for user's nbody simulation

C Interoperability Improvements

Interoperability Improvements

- Added 'isAnyCPtr()' and 'isExternClassType()' queries
- 'c_memcpy'/'c_memmove' allow 'c_void_ptr' arguments
- Added 'c_sizeof()' and 'c_memset()'
- 'writeln' can now print 'c_ptr' and 'c_void_ptr' variables
  - contributed by Nick Park

Other Library Improvements

Other Library Improvements

- **Renamed ‘Barrier’ module to ‘Barriers’**
  - To avoid having the 'Barrier' type share the same name as its module
- **Removed deprecated ‘RandomStream’ constructors**
- **Added waitAll() to ‘Futures’ module**
- **Added param/type overloads of getField() routines**
- **Added channel.lines() to iterate over lines in a channel**
- **Added file.getParentName() to ‘Path’ module**

Other Library Improvements

- Added datetime.ctime() to ‘DateTime’ module
- Added asciiToString() function
- Squashed entries with 0 as output from comm diagnostics

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