Chapel: Task-Based Communication in a Productive Language

Elliot Ronaghan, Chapel Team, Cray Inc.

SIAM PP18
March 8 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.
What is Chapel?

Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort

Goals:

- Support general parallel programming
  - “any parallel algorithm on any parallel hardware”
- Make parallel programming at scale far more productive
Chapel and Productivity

- Chapel strives to be…
  - as programmable as Python
  - as fast as Fortran
  - as scalable as MPI, SHMEM, or UPC
  - as portable as C
  - as flexible as C++
  - as fun as [your favorite programming language]
CLBG Cross-Language Summary
(Oct 2017 standings)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster

Copyright 2018 Cray Inc.
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)
CLBG Cross-Language Summary
(Oct 2017 standings)

Compressed Code Size (normalized to smallest entry)

Execution Time (normalized to fastest entry)

smaller

faster
Chapel Performance: HPC Benchmarks

**LCALS: Chapel vs. C + OpenMP**

- Shared memory performance competitive with hand-coded.
- Serial LCALS kernels: Chapel vs. g++.
- Parallel LCALS kernels: Chapel vs g++ w/ OMP.

**HPCC RA**

**HPCC RA Performance: Chapel vs. MPI**

** STREAM**

- HPCC Stream Triad: Chapel vs. MPI+OpenMP

** ISx**

- Isx Peformance: Chapel vs. MPI, SHMEM

** PRK**

- Stencil PRK Scalability

** Stencil**

- Stencil PRK Performance (weak scaling)

Nightly performance graphs online at: [https://chapel-lang.org/perf](https://chapel-lang.org/perf)
Chapel Performance: HPC Benchmarks

Local loop kernels

LCALS: Chapel vs. C + OpenMP
Shared memory performance competitive with hand-coded
Serial LCALS kernels: Chapel vs. g++
Parallel LCALS kernels: Chapel vs g++ w/ OMP

STREAM Triad

HPCC RA

PRK Stencil

HPCC Stream Triad: Chapel vs. MPI+OpenMP

Embarrassing/Pleasing Parallelism

Isx Peformance: Chapel vs. MPI, SHMEM

Bucket-Exchange Pattern

Stencil PRK Scalability

Global Random Updates

Nightly performance graphs online at: https://chapel-lang.org/perf
The Chapel Team at Cray (May 2017)

14 full-time employees + 2 summer interns + 2–4 GSoC students
Chapel Community Partners

(and several others…)

https://chapel-lang.org/collaborations.html
Plan for this talk

- Chapel by comparison: Random Access
- Runtime overview
- Performance optimizations enabled by runtime
Chapel by Comparison
HPCC Random Access

- Random Access (RA) benchmark
  - make random xor-updates to a distributed array of integers
  - stresses fine-grained communication (in its purest form)
  - benchmark allows up to 1% of updates to be missed/dropped
Random Access (GUPS) Kernel: MPI

/* Perform updates to main table. The scalar equivalent is: */
/* */
/* */
/* */
} while (i < SendCnt) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (i = 0; i < recvUpdates; i++) {
          inmsg = LocalRecvBuffer[bufferBase + i];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_TAG) {
        NumberReceiving =--;
        else
          MPI_Abort(MPI_COMM_WORLD, -1);
      }
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    } while (pendingUpdates > 0) {
      /* receive messages */
      do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
          if (status.MPI_TAG == UPDATE_TAG) {
            MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
            bufferBase = 0;
            for (i = 0; i < recvUpdates; i++) {
              inmsg = LocalRecvBuffer[bufferBase + i];
              LocalOffset = (inmsg & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
              HPCC_Table[LocalOffset] ^= inmsg;
            }
          } else if (status.MPI_TAG == FINISHED_TAG) {
            /* we got a done message. Thanks for playing... */
            NumberReceiving =--;
            else
              MPI_Abort(MPI_COMM_WORLD, -1);
          }
          MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        } while (have_done & & NumberReceiving > 0);
      } else {
        HPCC_InsertUpdate(Ran, WhichPe, Buckets);
        pendingUpdates++;
      } }++;
    } else {
      MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
      if (have_done) {
        outreq = MPI_REQUEST_NULL;
        if (params.finish_request == MPI_REQUEST_NULL) continue;
        else
          /* send our done messages */
          for (proc_count = 0; proc_count < tparams.NumProcs; ++proc_count) {
            if (params.finish_request == MPI профессиональное взаимодействие) {
              MPI_Request_free(outreq);
              MPI_Request_free(inreq);
              MPI_Wait(outreq, &status);
              MPI_Wait(inreq, &status);
            } else
              MPI_Abort(MPI_COMM_WORLD, -1);
          }
      } else if (params.finish_request == MPI профессиональное взаимодействие) {
        /* we got a done message. */
        NumberReceiving =--;
        else
          MPI_Abort(MPI_COMM_WORLD, -1);
      }
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    } MPI_Waitall(&params.NumProcs, tparams.finish_rq + proc_count);
  } /* Finish everyone else up... */
  while (NumberReceiving > 0) {
    MPI_Wait(&inreq, &status);
    if (status.MPI_TAG == UPDATE_TAG) {
      MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
      bufferBase = 0;
      for (i = 0; i < recvUpdates; i++) {
        inmsg = LocalRecvBuffer[bufferBase + i];
        LocalOffset = (inmsg & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
        HPCC_Table[LocalOffset] ^= inmsg;
      }
    } else if (status.MPI_TAG == FINISHED_TAG) {
      /* we got a done message. Thanks for playing... */
      NumberReceiving =--;
      else
        MPI_Abort(MPI_COMM_WORLD, -1);
    }
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
  } MPI_Waitall(&params.NumProcs, tparams.finish_rq + tparams.finish_statuses);
/* Perform updates to main table. The scalar equivalent is: */

* for (i=0; i<NUPDATE; i++) {
* Ran = (Ran << 1) ^ (((s64Int) Ran < ZERO64B ? POLY : 0);
* Table[Ran & (TABSIZE-1)] ^= Ran;
* }
*/

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (have_done & NumberReceiving > 0);
if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : 0);
    Table[Ran & (TABSIZE-1)] ^= Ran;
}
MPI_Waitall(tparams.NumProcs, tparams.finish_req,
&status, tparams.dtype64, &inmsg,
&tparams.finish_statuses);

foreach (b in Buckets) {
    for (i=0; i<NUPDATE; i++) {
        for (j=0; j < tparams.TableSize - 1; j++) {
            inmsg = tparams.Table[b][j];
            // Update Table[b][j]
        }
    }
}

// we got a done message. Thanks for playing... */
NumberReceiving--;
} else {
    MPI_Abort(MPI_COMM_WORLD, -1);
}
MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (have_done & NumberReceiving > 0);

/* Perform updates to main table. The scalar equivalent is: */

* for (i=0; i<NUPDATE; i++) {
* Ran = (Ran << 1) ^ (((s64Int) Ran < ZERO64B ? POLY : 0);
* Table[Ran & (TABSIZE-1)] ^= Ran;
* }
*/

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (have_done & NumberReceiving > 0);
if (have_done) {
    if (status.MPI_TAG == UPDATE_TAG) {
        bufferBase = 0;
        for (j=0; j < recvUpdates; j++) {
            inmsg = LocalRecvBuffer[bufferBase+j];
            LocalOffset = (inmsg & tparams.TableSize - 1) - tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= inmsg;
        }
    } else if (status.MPI_TAG == FINISHED_TAG) {
        // we got a done message. Thanks for playing... */
        NumberReceiving--;
    } else {
        MPI_Abort(MPI_COMM_WORLD, -1);
    }
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    while (have_done & NumberReceiving > 0);

/* Perform updates to main table. The scalar equivalent is: */

* for (i=0; i<NUPDATE; i++) {
* Ran = (Ran << 1) ^ (((s64Int) Ran < ZERO64B ? POLY : 0);
* Table[Ran & (TABSIZE-1)] ^= Ran;
* }
*/

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (have_done & NumberReceiving > 0);
if (have_done) {
    if (status.MPI_TAG == UPDATE_TAG) {
        bufferBase = 0;
        for (j=0; j < recvUpdates; j++) {
            inmsg = LocalRecvBuffer[bufferBase+j];
            LocalOffset = (inmsg & tparams.TableSize - 1) - tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= inmsg;
        }
    } else if (status.MPI_TAG == FINISHED_TAG) {
        // we got a done message. Thanks for playing... */
        NumberReceiving--;
    } else {
        MPI_Abort(MPI_COMM_WORLD, -1);
    }
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    while (have_done & NumberReceiving > 0);

else if (have_done) {
    if (status.MPI_TAG == UPDATE_TAG) {
        bufferBase = 0;
        for (j=0; j < recvUpdates; j++) {
            inmsg = LocalRecvBuffer[bufferBase+j];
            LocalOffset = (inmsg & tparams.TableSize - 1) - tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= inmsg;
        }
    } else if (status.MPI_TAG == FINISHED_TAG) {
        // we got a done message. Thanks for playing... */
        NumberReceiving--;
    } else {
        MPI_Abort(MPI_COMM_WORLD, -1);
    }
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    while (have_done & NumberReceiving > 0);

/* MPI comment */
/* Perform updates to main table. The scalar equivalent is:
 * for (i=0; i<NUPDATE; i++) {
 *     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *     Table[Ran & (TABSIZETAG)] ^= Ran;
 * }
 */

forall (_, r) in zip(Updates, RAStream()) do
    T[r & indexMask] ^= r;
for all (i, r) in zip(Updates, RAStream()) do
  T[r & indexMask] ^= r;

Copyright 2018 Cray Inc.
for all (__, r) in zip(Updates, RAStream()) do
  T[r & indexMask] ^= r;

forall (__, r) in zip(Updates, RAStream()) do
  on TableDist.idxToLocale[r & indexMask] do
    T[r & indexMask] ^= r;
Random Access (GUPS) Kernel: Chapel

for all (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask] ^= r;

forall (_, r) in zip(Updates, RAStream())
  on TableDist.idxToLocale[r & indexMask] do
    T[r & indexMask] ^= r;

forall (_, r) in zip(Updates, RAStream())
  T[r & indexMask].xor(r);

Hello world

Thanks for playing... */

/* receive messages */
while (have_done > 0) {
  MPI_Test(&status, tparams.dtype64, &recvUpdates);
  if (have_done) {
    if (status.MPI_TAG == UPDATE_TAG) {
      MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
      if (recvUpdates > 0) {
        for (j=0; j < recvUpdates; j++) {
          inmsg = LocalRecvBuffer[bufferBase + j] ^=
            HPCC_Table[LocalOffset] ^=
              (Ran, 0, tparams.dtype64, &params;
        } else if (status.MPI_TAG == FINISHED_TAG) {
          /* we got a done message. Thanks for playing... */
          send our done messages */
          for (proc_count = 0; proc_count < tparams.NumProcs; ++proc_count) {
            if (proc_count == tparams.MyProc) {
              tparams.finish_statuses[proc_count] = FINISHED_TAG;
            }
          }
        } else if (status.MPI_TAG == REQUEST_TAG) {
          if (params.send_req[params.MyProc] == FINISHED_TAG) {
            MPI_Waitall(tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
          }
        } else if (status.MPI_TAG == UPDATE_TAG) {
          /* send remaining updates in buckets */
          if (IndexMask == 0) {
            MPI_Waitall(tparams.NumProcs, tparams.finish_statuses);
          }
          MPI_Request_NULL; continue;
        } else if (status.MPI_TAG == NONE) {
          /* send garbage */
          MPI_Request_NULL; continue;
        }
      }
    } else if (status.MPI_TAG == FINISHED_TAG) {
      /* we got a done message. Thanks for playing... */
      send our done messages */
      for (proc_count = 0; proc_count < tparams.NumProcs; ++proc_count) {
        if (proc_count == tparams.MyProc) {
          tparams.finish_statuses[proc_count] = FINISHED_TAG;
        }
      }
    } else if (status.MPI_TAG == REQUEST_TAG) {
      if (params.send_req[params.MyProc] == FINISHED_TAG) {
        MPI_Waitall(tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
      }
    } else if (status.MPI_TAG == UPDATE_TAG) {
      /* send remaining updates in buckets */
      if (IndexMask == 0) {
        MPI_Waitall(tparams.NumProcs, tparams.finish_statuses);
      }
      MPI_Request_NULL; continue;
    } else if (status.MPI_TAG == NONE) {
      /* send garbage */
      MPI_Request_NULL; continue;
    }
  }
}

/* Finish everyone else up... */
for (proc_count = 0; proc_count < tparams.NumProcs; ++proc_count) {
  if (proc_count == tparams.MyProc) {
    tparams.finish_statuses[proc_count] = FINISHED_TAG;
  }
}

for (j=0; j < proc_count; j++) {
  inmsg = LocalRecvBuffer[bufferBase + j] ^=
    HPCC_Table[LocalOffset] ^=
      (Ran, 0, tparams.dtype64, &params;

HPCC RA Performance: Chapel vs. MPI

Performance of RA
(Chapel vs MPI)

GUP/s vs Locales

Chapel vs MPI
Reference no-bucketing
Chapel
Chapel Runtime
Compiling Chapel

Chapel Source Code → chpl → Chapel Executable

Standard Modules (in Chapel)
Chapel Compilation Architecture

- Chapel Source Code
- Chapel-to-C Compiler
- Generated C Code
- Standard C Compiler & Linker
- Chapel Executable
- Standard Modules (in Chapel)
- Internal Modules (in Chapel)
- Runtime Support Library (in C)
  - Tasks/Threads
  - Communication
  - Memory
  -...

Chapel Compiler
Chapel Compilation Architecture

- Chapel Source Code
- Chapel-to-C Compiler
- Generated C Code
- Standard C Compiler & Linker
- Chapel Executable

- Standard Modules (in Chapel)
- Internal Modules (in Chapel)

Runtime Support Library (in C)
- Tasks/Threads
- Communication
- Memory
- ...
Chapel Runtime

- Lowest level of Chapel software stack
- Supports language concepts and program activities
  - Task creation, communication, memory allocation
- Composed of layers
  - Standardized interfaces
  - Interchangeable implementations
Chapel Runtime Organization

Chapel Runtime Support Library

- Communication
- Tasking
- Memory
- Launchers
- QIO
- Timers
- Standard

Standard and third-party libraries
Runtime Tasking Layer

Chapel Runtime Support Library

Tasking

- fifo
- Qthreads Tasks (Sandia)
- Massive-Threads (U Tokyo)

Synchronization

POSIX Threads

pthreads
Runtime Tasking Layer

- **Supports parallelism**
- **Operations**
  - Create a group of tasks
  - Start a “moved” task
    - Start a remote task “moved” by the comm layer

- Chapel Runtime Support Library
- fifo
- Qthreads Tasks (Sandia)
- Massive Threads (U Tokyo)
- pthreads
- POSIX Threads
Runtime Tasking Layer

Chapel Runtime Support Library

Qthreads Overview

- fifo
- pthreads
- Qthreads Tasks (Sandia)
- Massive-Threads (U Tokyo)

POSIX Threads

Synchronization
Qthreads Overview

- Lightweight, locality-aware tasking library
  - cooperative scheduling
  - qthreads are entirely in user space
    - extremely fast task creation and switching
  - designed to be highly concurrent
    - run millions of qthreads, limited only by available memory
  - locality-aware
  - multiple scheduler options
    - From simple fifo queues to advanced work-stealing schedulers
Runtime Communication Layer

Chapel Runtime Support Library

Communication

ugi

GASNet (Berkeley)
Runtime Communication Layer

- **Supports Communication**
  - gets, puts, remote-task-creation

- **Works with tasking layer (through API)**
  - allows arbitrary comm/compute overlap

```c
chpl_comm_put(...){
    done = do_remote_put(...);
    while (!complete(&done)) {
        chpl_task_yield(); // yield while waiting for network
    }
}
```
Communication + Tasking Overview

● “Unified” runtime permits many optimizations
  ● standardized APIs prevent unnecessary/harmful coupling

● e.g. allows communication and computation overlap
  ● in a trivial to implement manner
  ● task switching in user-space makes this fast
  ● cooperative tasking minimizes overhead for creating many tasks
Comm/compute overlap with RA
HPCC RA Performance: Chapel vs. MPI

Performance of RA
(Chapel vs MPI)

Locales

GUP/s

Reference no-bucketing
Chapel

better

Copyright 2018 Cray Inc.
Update Dist Declaration

UpdateDist = new dmap(new Block(boundingBox={0..N_U-1}));
HPCC RA Performance: Chapel vs. MPI

Update Dist Declaration

UpdateDist = new dmap(new Block(boundingBox={0..N_U-1},
  tasksPerLocale=2*here.maxTaskPar));

Performance of RA
(Chapel vs MPI)

Locales

GUP/s

0 0.5 1 1.5 2 2.5

16 32 64 128 256

Reference no-bucketing
Chapel
Chapel oversubscribed
Wrapping Up
Summary

- Chapel is a productive parallel programming language
  - productivity enabled by rich tasking and communication runtime

- **Flexible, but cohesive runtime enables optimizations**
  - e.g. comm/compute overlap
  - future avenues: distributed work stealing, comm aggregation

- Chapel performance can match C+MPI+OpenMP
  - with improvements in readability, writability, code size
Chapel Resources
Chapel Central: https://chapel-lang.org/

The Chapel Parallel Programming Language

What is Chapel?
Chapel is a modern programming language that is...
- scalable: supports locality-oriented features for distributed memory systems
- portable: runs on laptops, clusters, the cloud, and HPC systems
- productive: designed with programmability and performance in mind
- parallel: contains first-class concepts for concurrent and parallel computation

New to Chapel?
As an introduction to Chapel, you may want to...
- read a blog article or book chapter
- watch an overview talk or browse its slides
- download the release
- browse sample programs
- view other resources to learn how to trivially write distributed programs like this:

```plaintext
forall i in (1:n) dropped (n+1)
    do writeline("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What's Hot?
- Chapel 1.16 is now available—download a copy today!
- The CHIUW 2018 call for participation is now available!
- A recent Cray blog post reports on highlights from CHIUW 2017.
- Chapel is now one of the supported languages on Try It Online!
- Watch talks from ACCU 2017, CHIUW 2017, and ATPESC 2016 on YouTube.
- Browse slides from PADAL, EAGE, EMBRACE, ACCU, and other recent talks.
- See also: What's New?
How to Track Chapel

http://facebook.com/ChapelLanguage
http://twitter.com/ChapelLanguage
https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/chapel-announce@lists.sourceforge.net
Suggested Reading (healthy attention spans)

Chapel chapter from *Programming Models for Parallel Computing*

- a detailed overview of Chapel’s history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is now also available online

Other Chapel papers/publications available at [https://chapel-lang.org/papers.html](https://chapel-lang.org/papers.html)
Suggested Reading (short attention spans)

- a run-down of recent events

- a short-and-sweet introduction to Chapel

**Six Ways to Say “Hello” in Chapel** (parts 1, 2, 3), Cray Blog, Sep-Oct 2015.
- a series of articles illustrating the basics of parallelism and locality in Chapel

**Why Chapel?** (parts 1, 2, 3), Cray Blog, Jun-Oct 2014.
- a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges

- a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages
Chapel StackOverflow and GitHub Issues
Where to..

Submit bug reports:
GitHub issues for chapel-lang/chapel: public bug forum
chapel_bugs@cray.com: for reporting non-public bugs

Ask User-Oriented Questions:
StackOverflow: when appropriate / other users might care
#chapel-users (irc.freenode.net): user-oriented IRC channel
chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development
chapel-developers@lists.sourceforge.net: developer discussions
#chapel-developers (irc.freenode.net): developer-oriented IRC channel

Discuss Chapel’s use in education
chapel-education@lists.sourceforge.net: educator discussions

Directly contact Chapel team at Cray: chapel_info@cray.com
Questions?
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.