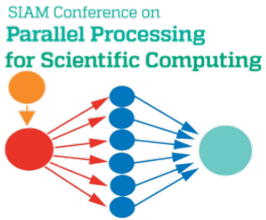
Chapel: Task-Based Communication in a Productive Language

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

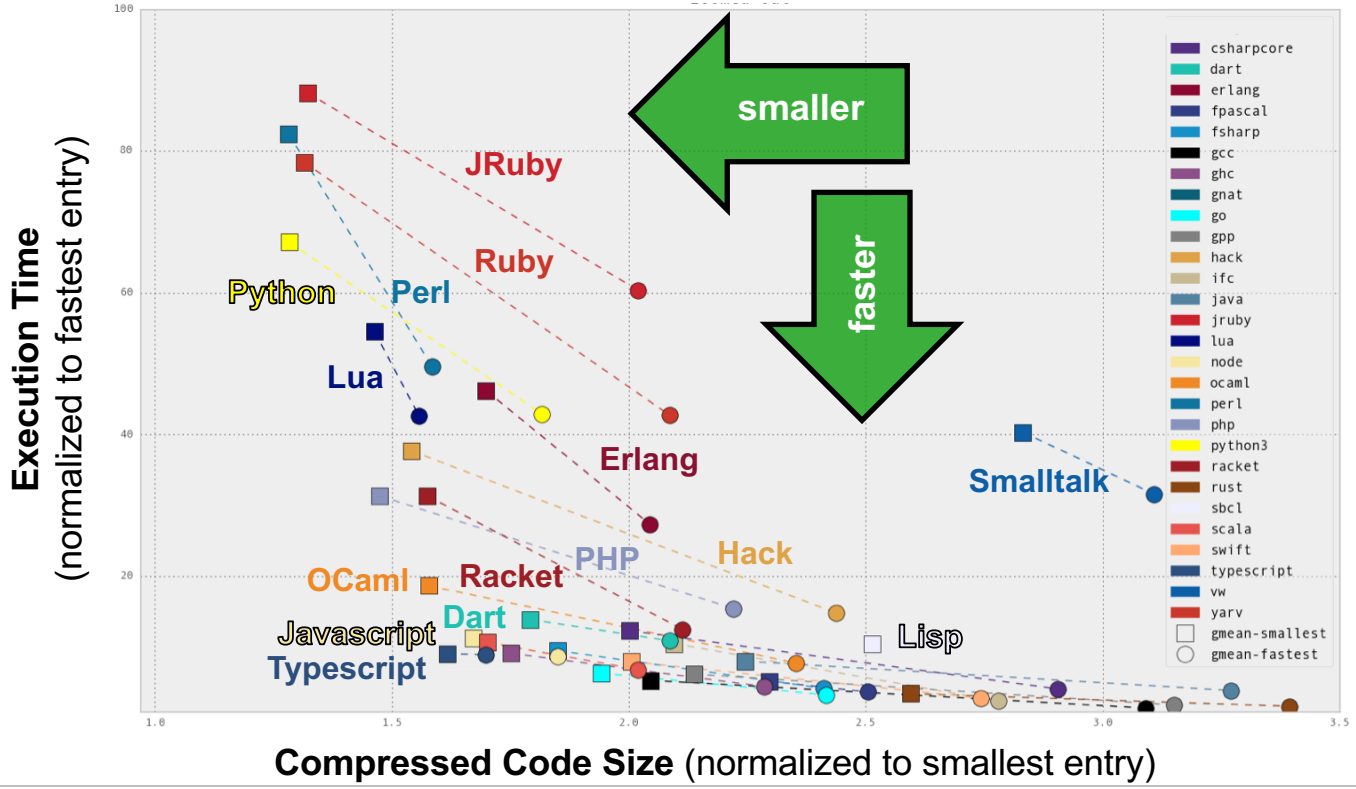
Goals:

- Support general parallel programming
  - "any parallel algorithm on any parallel hardware"
- Make parallel programming at scale far more productive

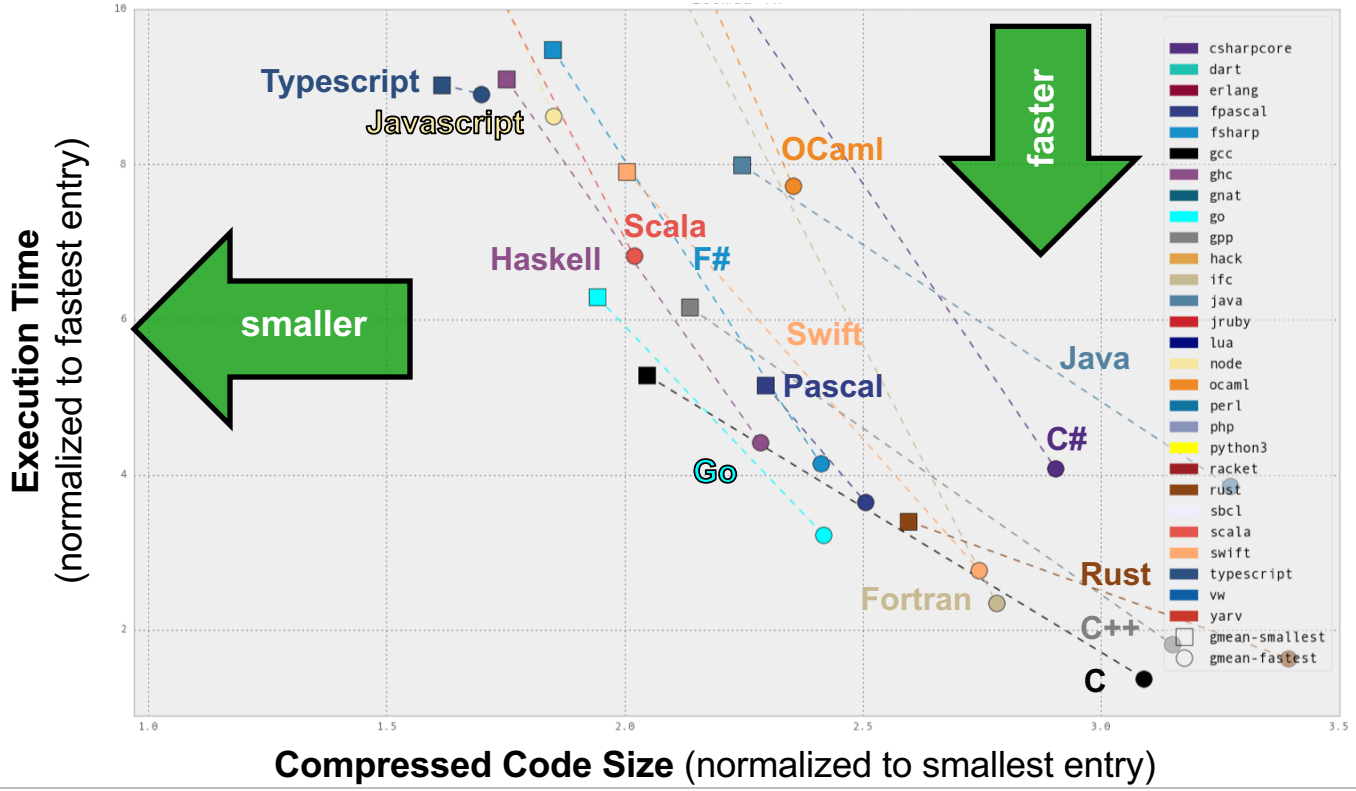
Chapel and Productivity

- **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]

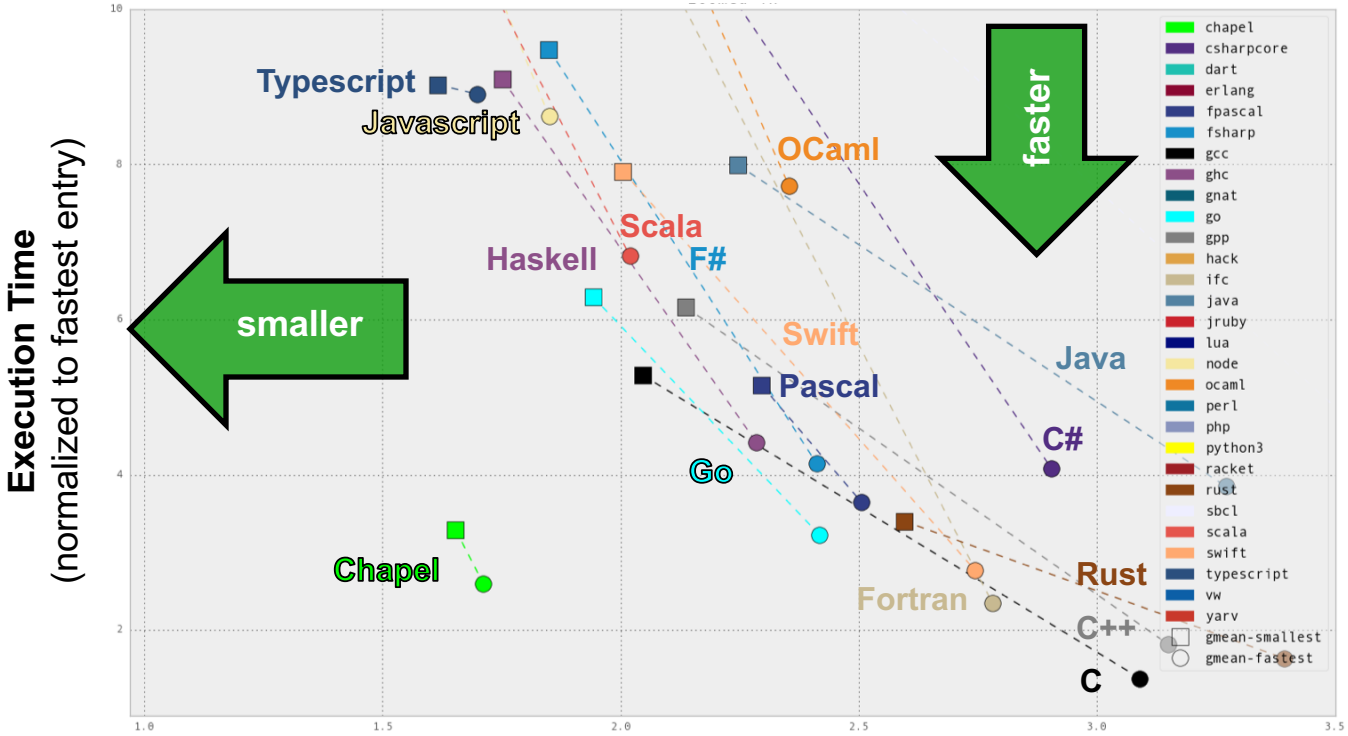
CLBG Cross-Language Summary
(Oct 2017 standings)



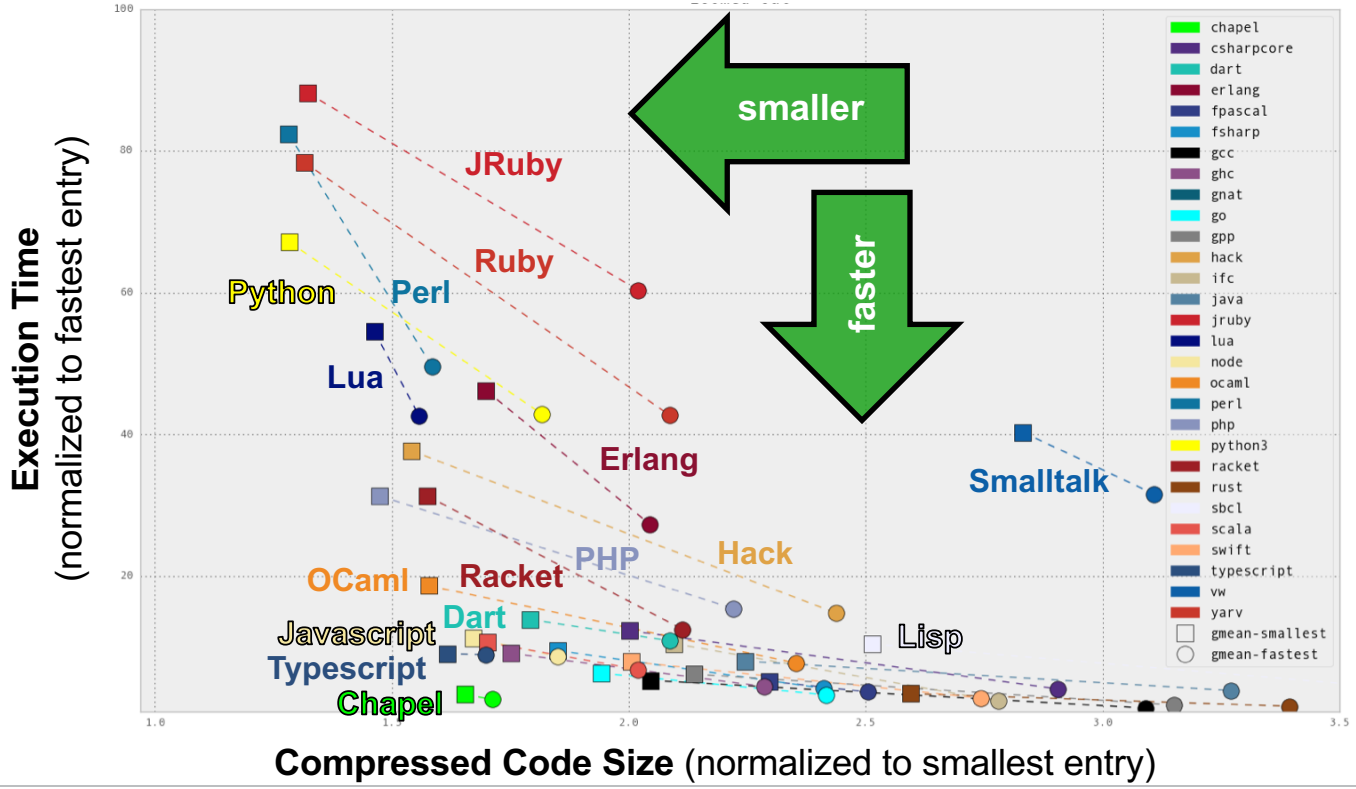
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)



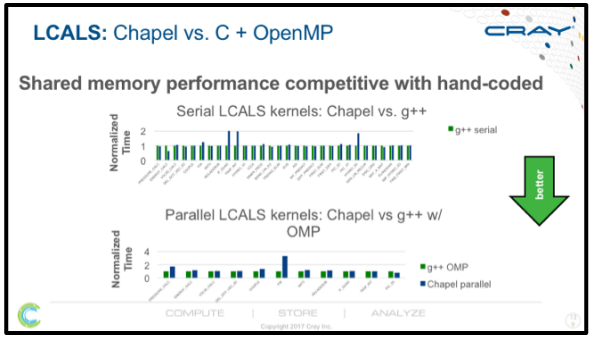
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)



CLBG Cross-Language Summary
(Oct 2017 standings)

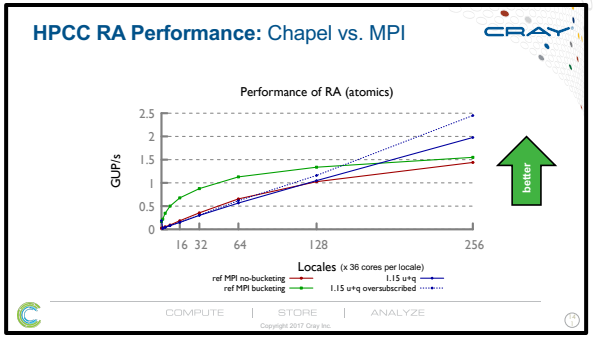


Chapel Performance: HPC Benchmarks



LCALS

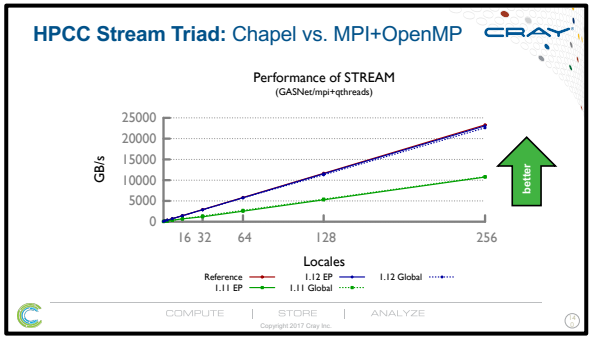
HPCC RA

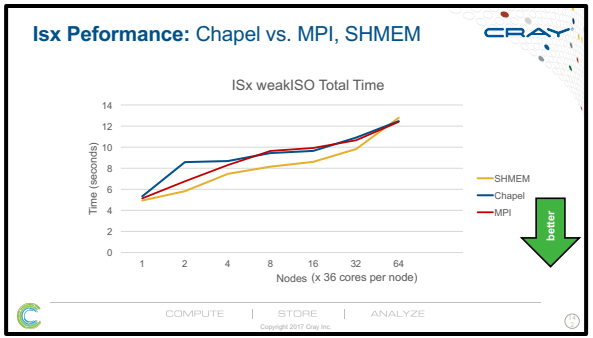


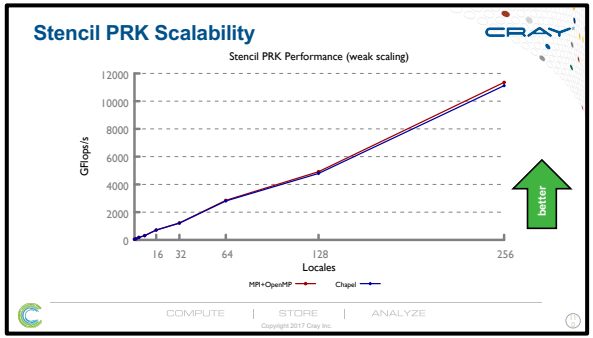
STREAM Triad

ISx

PRK Stencil

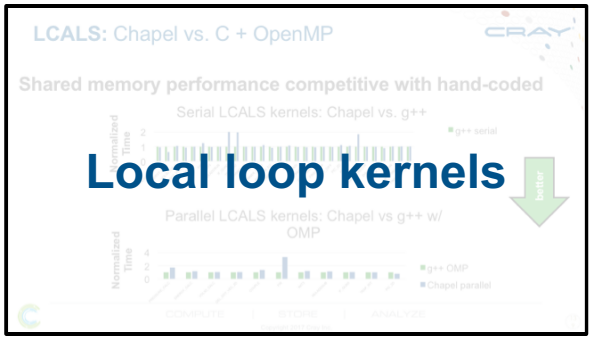




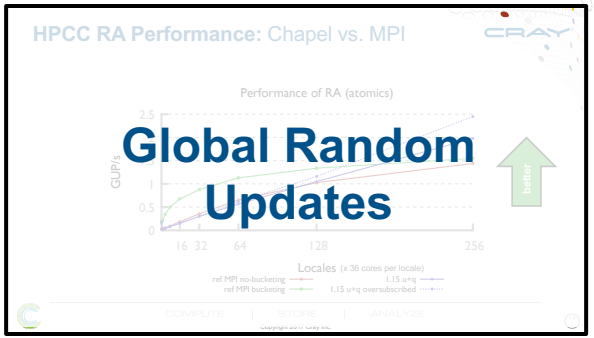


Nightly performance graphs online at: https://chapel-lang.org/perf

Chapel Performance: HPC Benchmarks

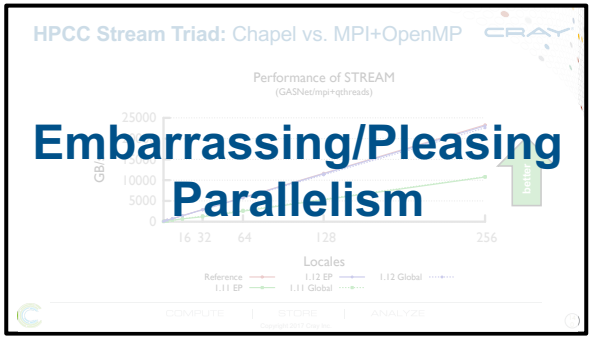


LCALS

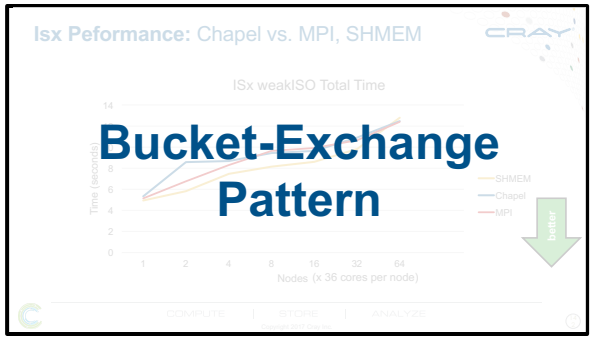


HPCC RA

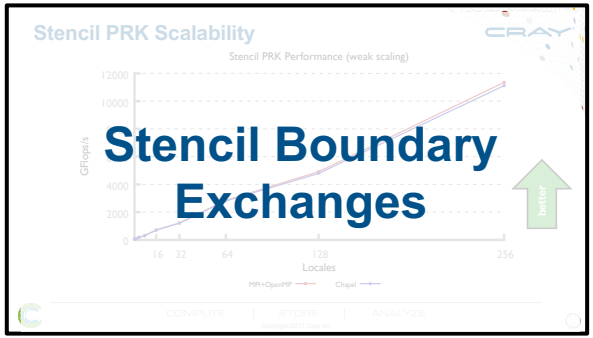
STREAM Triad



ISx



PRK Stencil



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

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
        MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
 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 (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) {
        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);
  if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
    GlobalOffset = Ran & (tparams.TableSize-1);
    if ( GlobalOffset < tparams.Top)
      WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
    else
      WhichPe = ( (GlobalOffset - tparams.Remainder) / tparams.MinLocalTableSize );
    if (WhichPe == tparams.MyProc) {
      LocalOffset = (Ran & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
      HPCC_Table[LocalOffset] ^= Ran;
    } else {
      HPCC_InsertUpdate(Ran, WhichPe, Buckets);
      pendingUpdates++;
    }
    i++;
  }
  else {
    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
      outreq = MPI_REQUEST_NULL;
      pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize, &peUpdates);
      MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe, UPDATE_TAG,
            MPI_COMM_WORLD, &outreq);
      pendingUpdates -= peUpdates;
    }
  }
}
/* send remaining updates in buckets */
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 (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_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
      outreq = MPI_REQUEST_NULL;
      pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize, &peUpdates);
      MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe, UPDATE_TAG
            MPI_COMM_WORLD, &outreq);
      pendingUpdates -= peUpdates;
    }
  }
  /* send our done messages */
  for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
    if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
MPI_REQUEST_NULL; continue; }
    /* send garbage - who cares, no one will look at it */
    MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
          MPI_COMM_WORLD, tparams.finish_req + 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 (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);
  }

  MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

Random Access (GUPS) Kernel: Chapel

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 & (TABSIZE-1)] ^= Ran;
 *      }
 */
```

Random Access (GUPS) Kernel: Chapel

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 & (TABSIZE-1)] ^= Ran;
 *      }
 */
```

Chapel Code

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

Random Access (GUPS) Kernel: Chapel

Chapel RMO

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

Random Access (GUPS) Kernel: Chapel

Chapel RMO

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

Chapel On-stmt

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

Random Access (GUPS) Kernel: Chapel

Chapel RMO

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

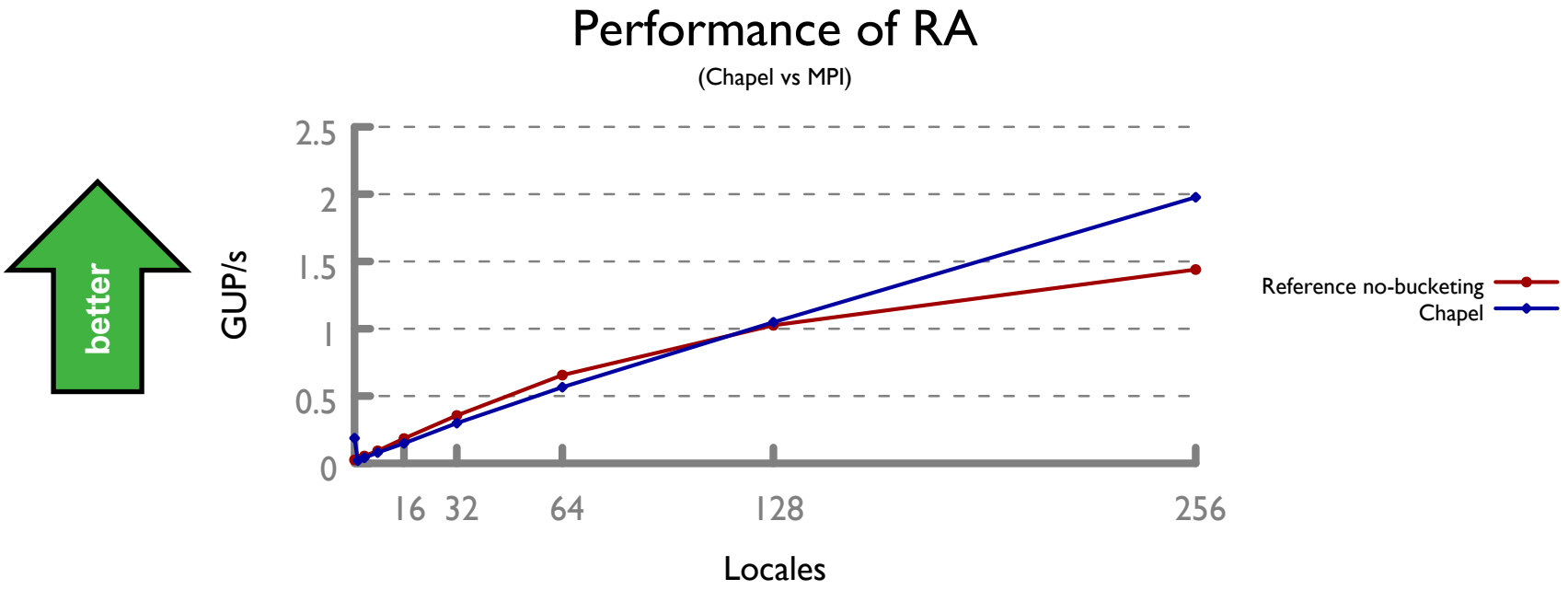
Chapel On-stmt

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

Chapel Atomic

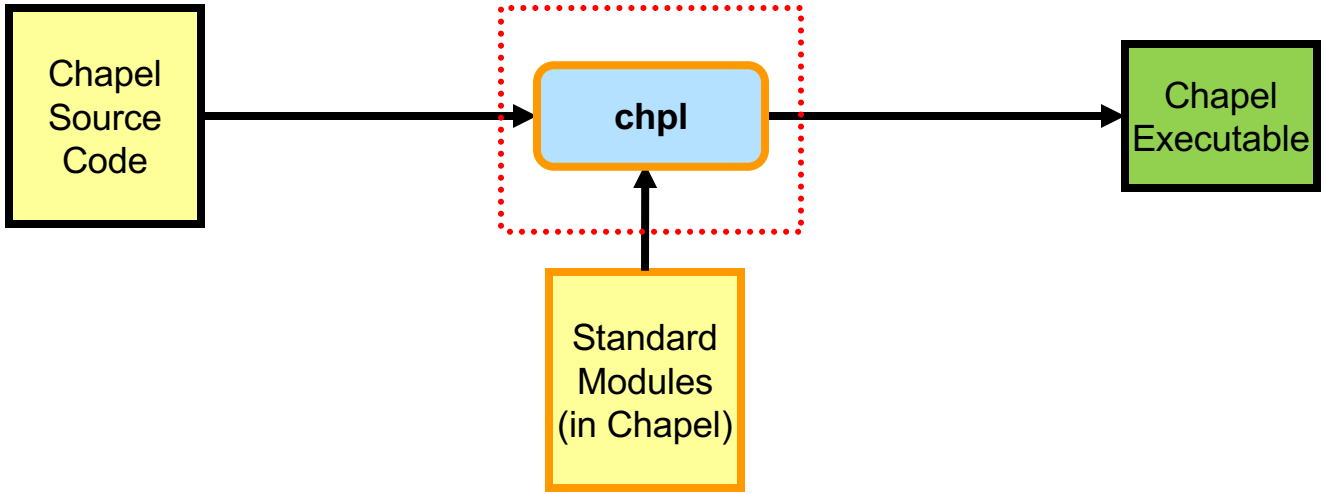
```
forall (_, r) in zip(Updates, RAStream()) do
    T[r & indexMask].xor(r);
```

HPCC RA Performance: Chapel vs. MPI

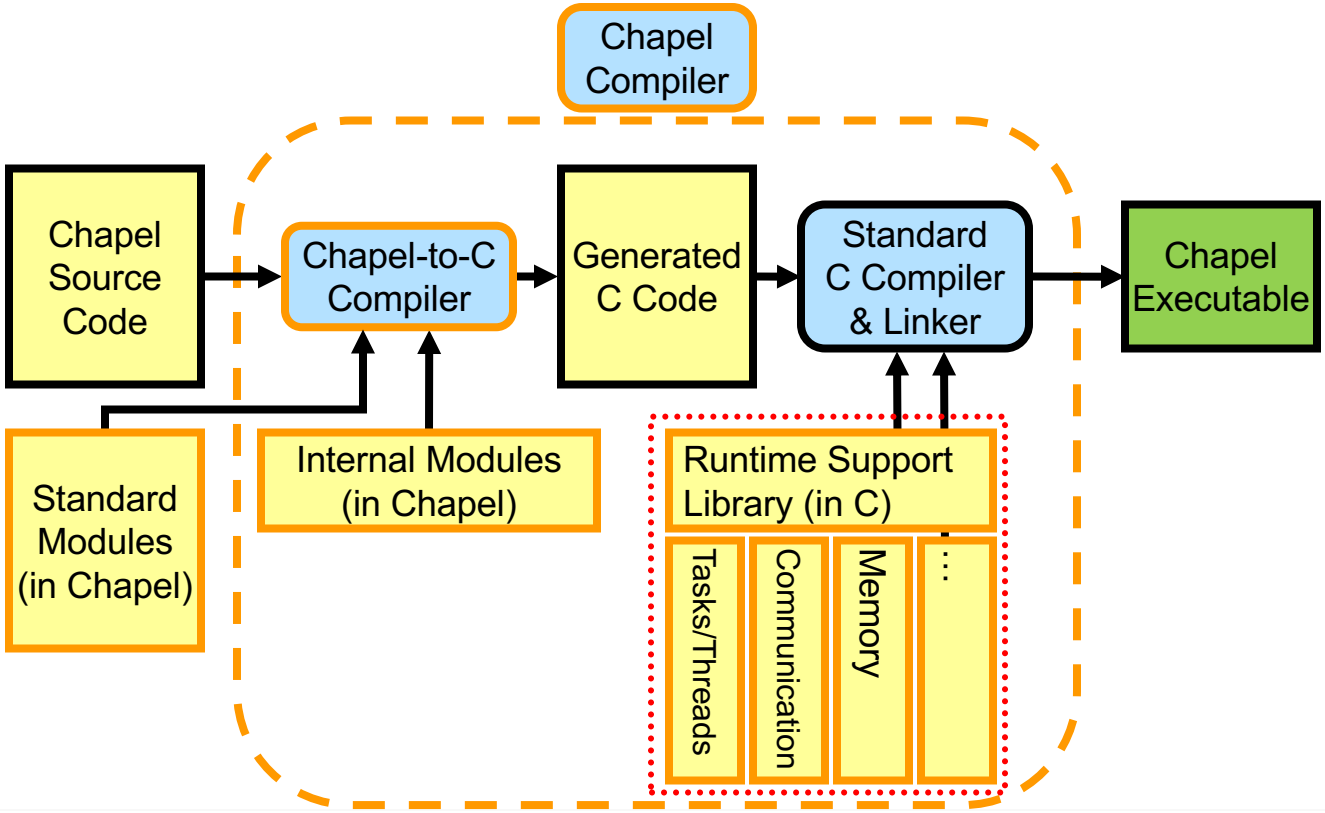


Chapel Runtime

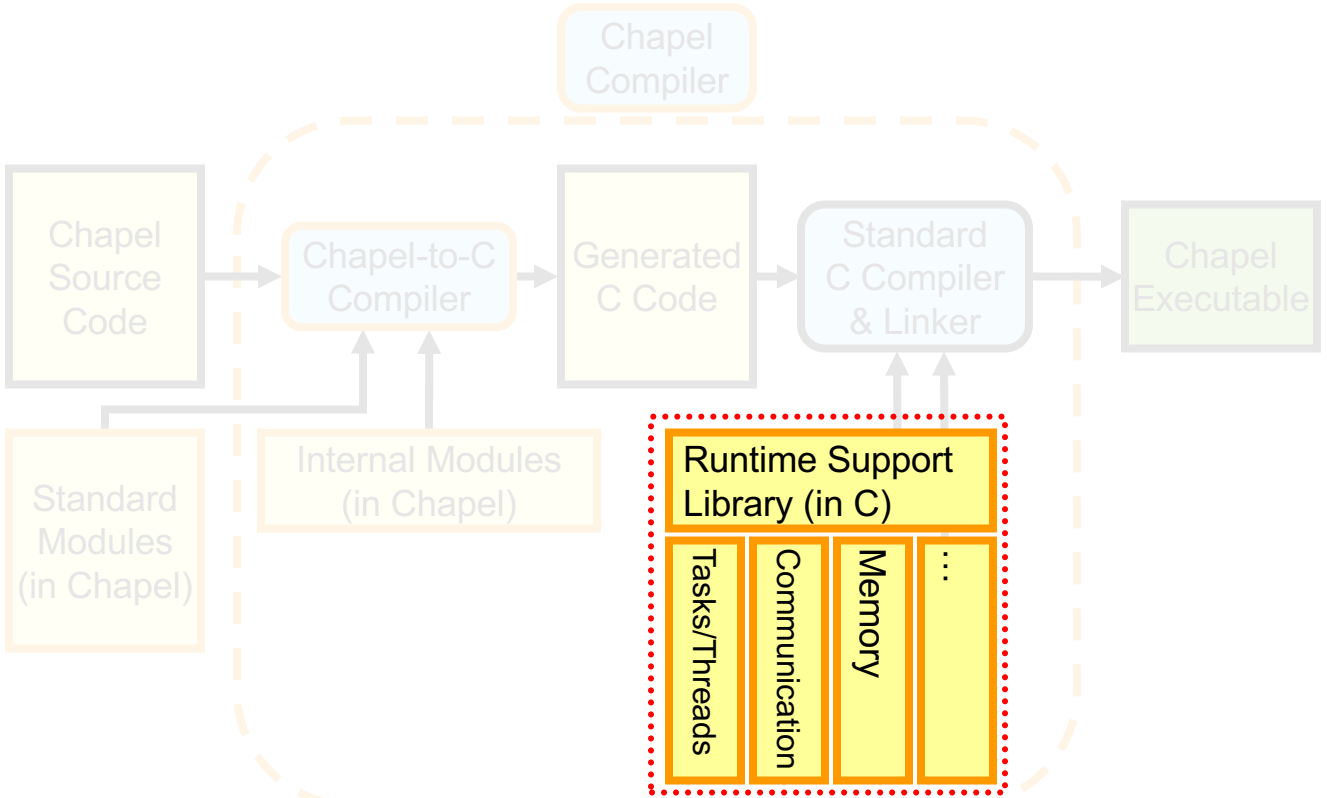
Compiling Chapel



Chapel Compilation Architecture



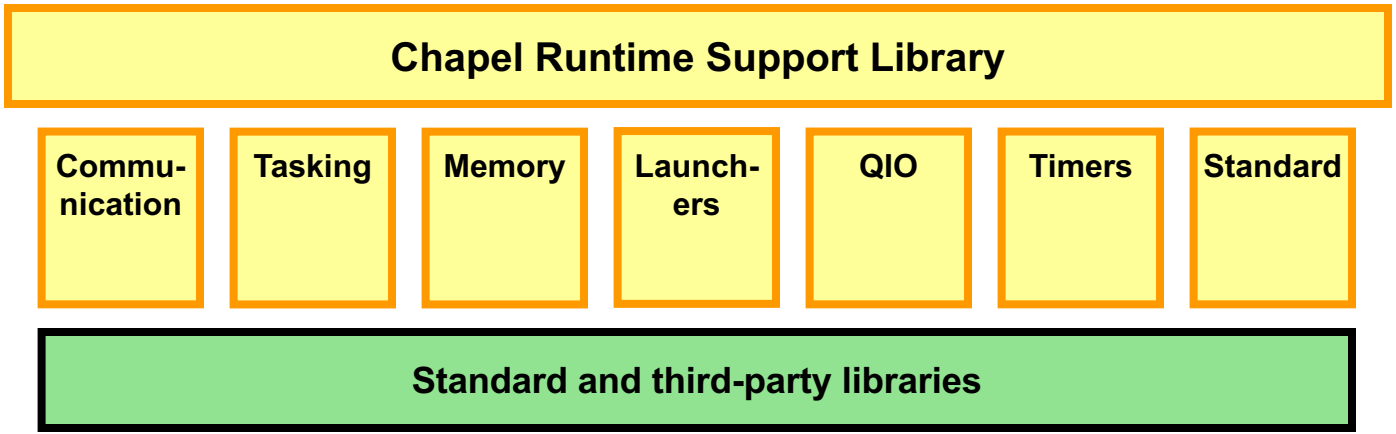
Chapel Compilation Architecture



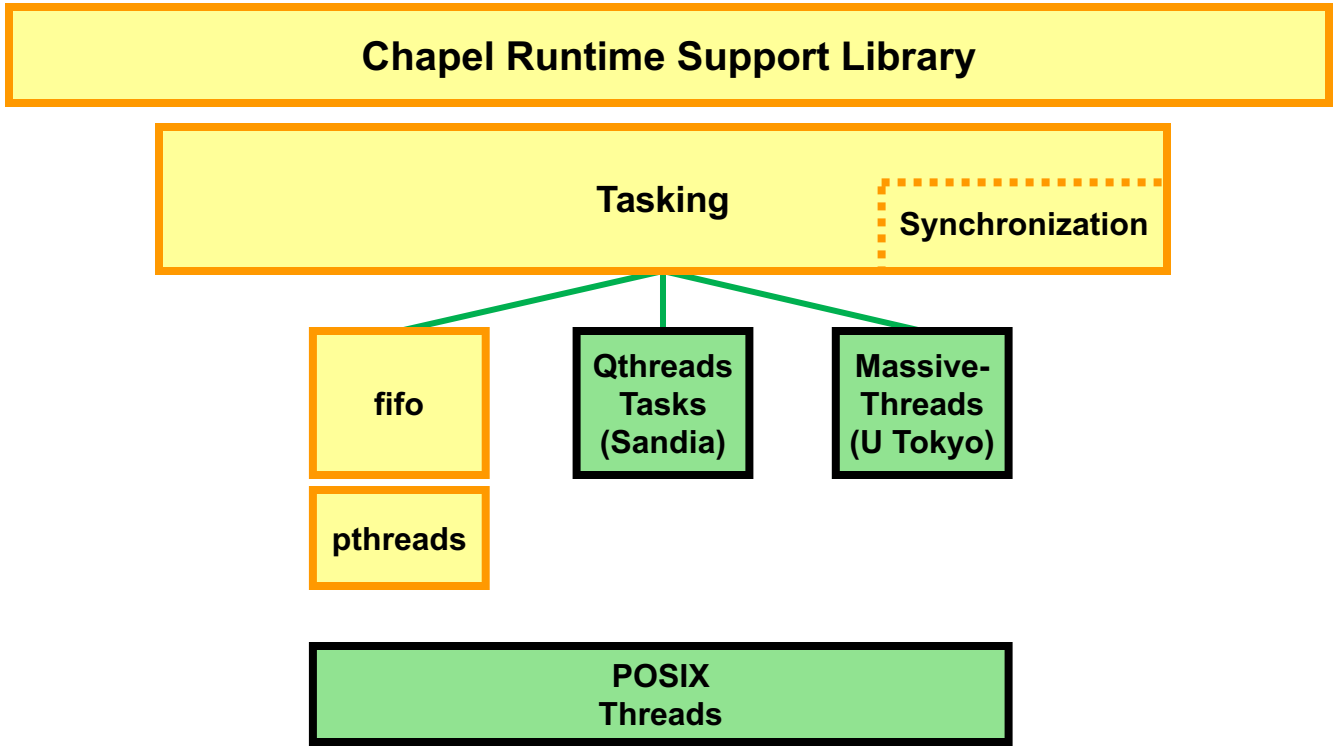
Chapel Runtime

- **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



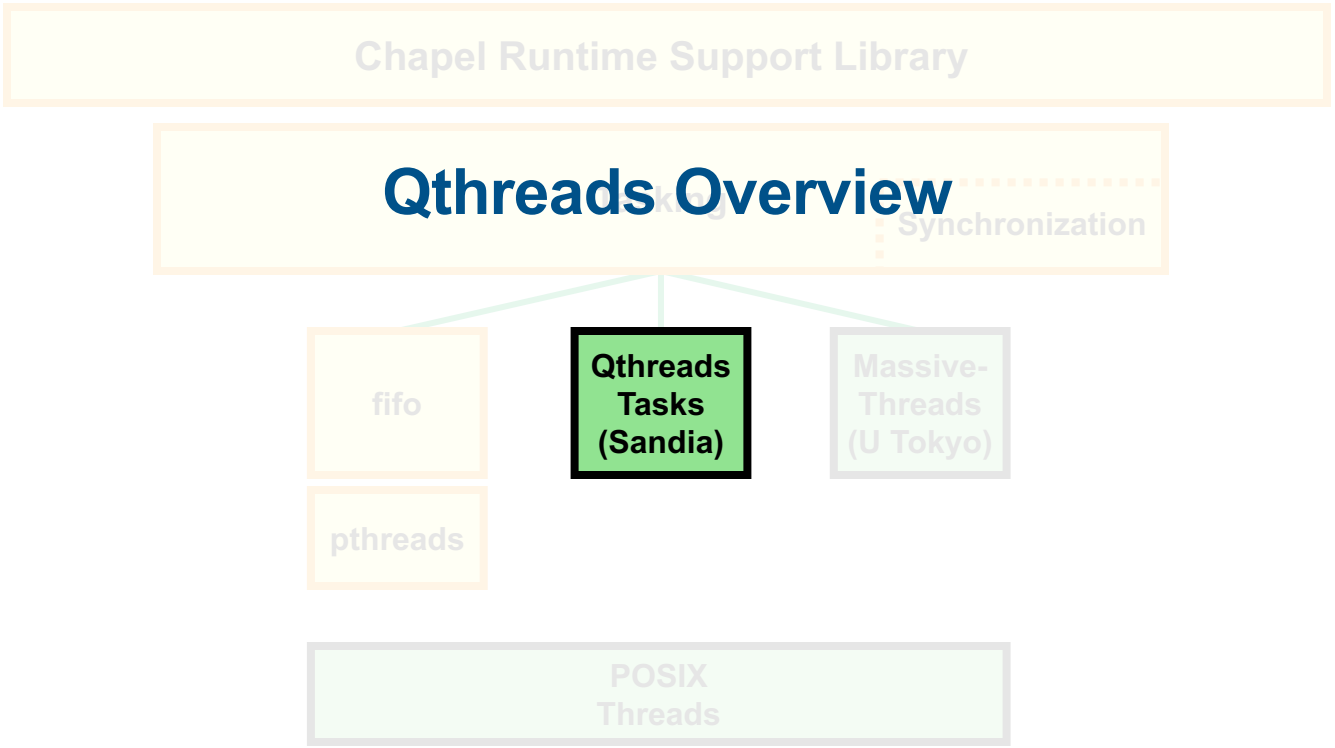
Runtime Tasking Layer



Runtime Tasking Layer

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

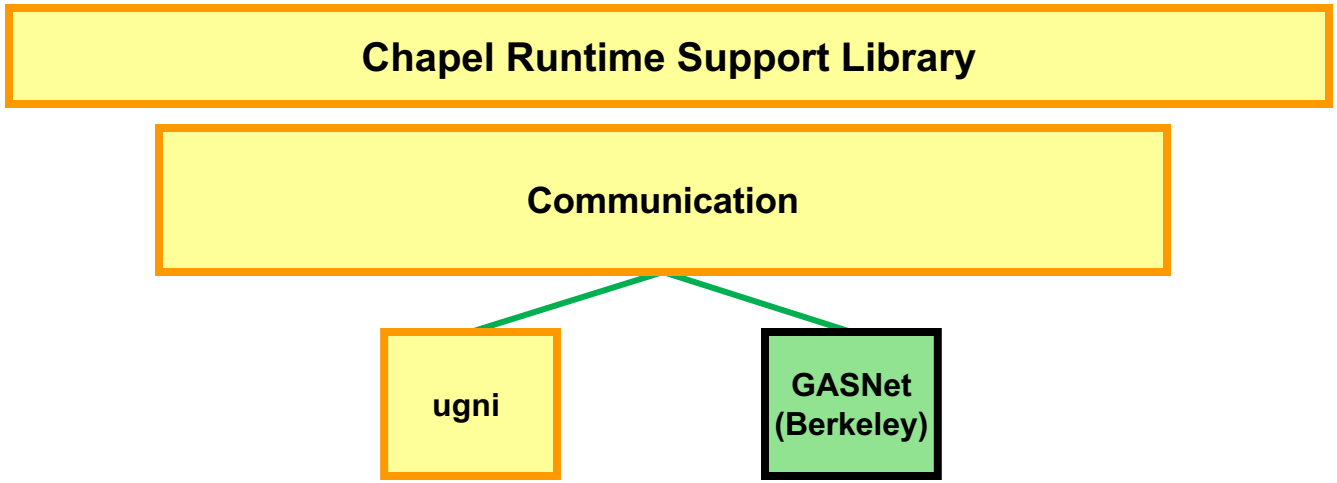
Runtime Tasking Layer



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



Runtime Communication Layer

- **Supports Communication**
  - gets, puts, remote-task-creation
- **Works with tasking layer (through API)**
  - allows arbitrary comm/compute overlap

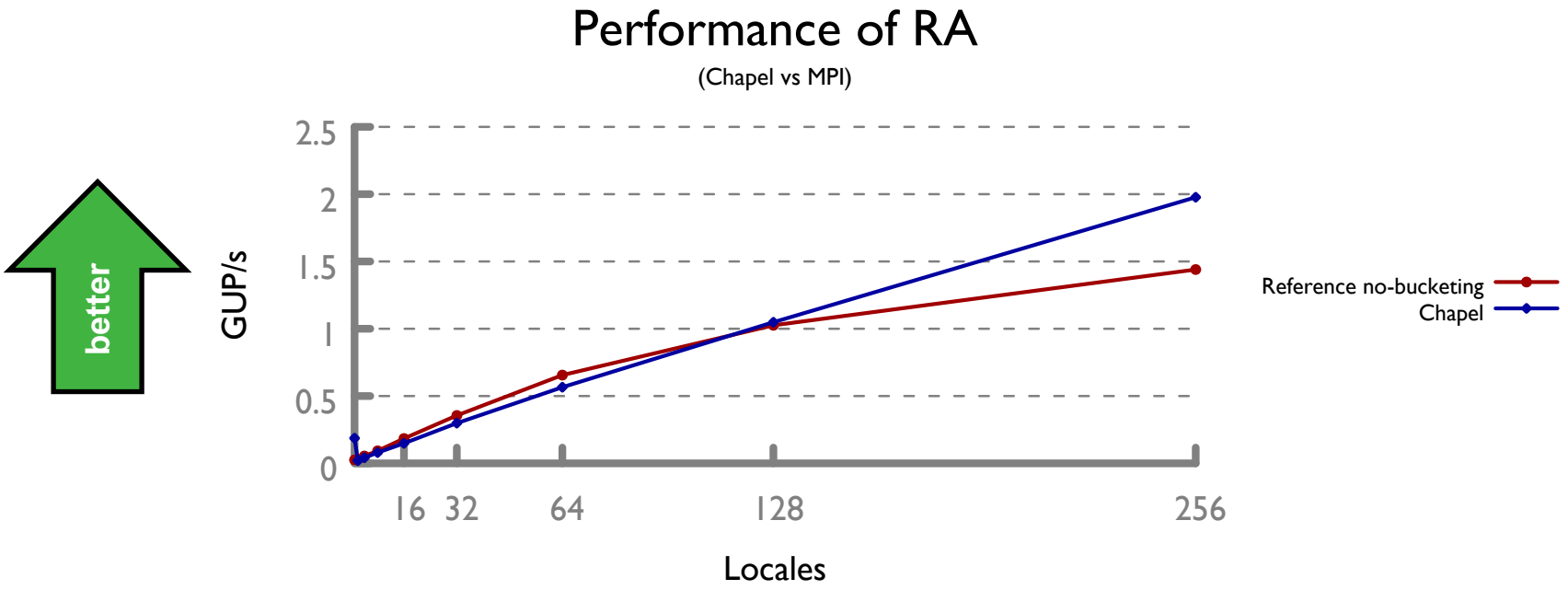
```
chpl_comm_put(…){
  done = do_remote_put(…);
  while (!complete(&done)) {
    chpl_task_yield(); // yield while waiting for network
  }
}
```

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

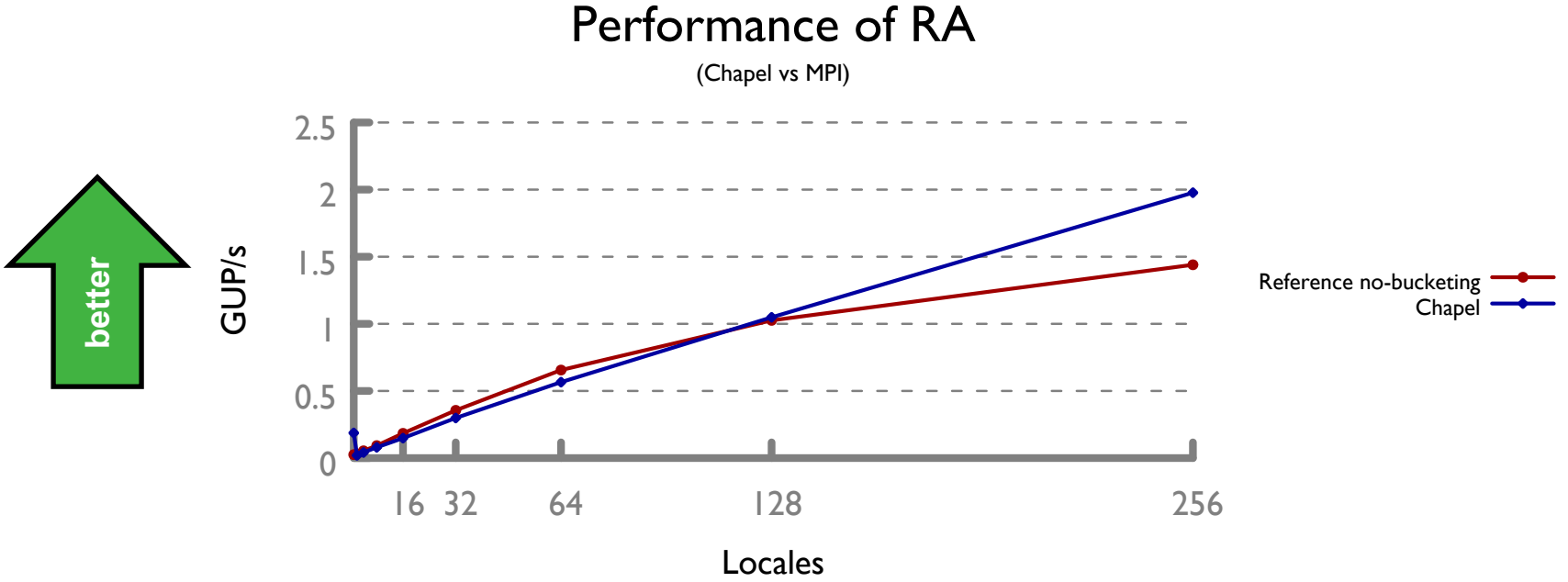
HPCC RA Performance: Chapel vs. MPI



HPCC RA Performance: Chapel vs. MPI

Update Dist Declaration

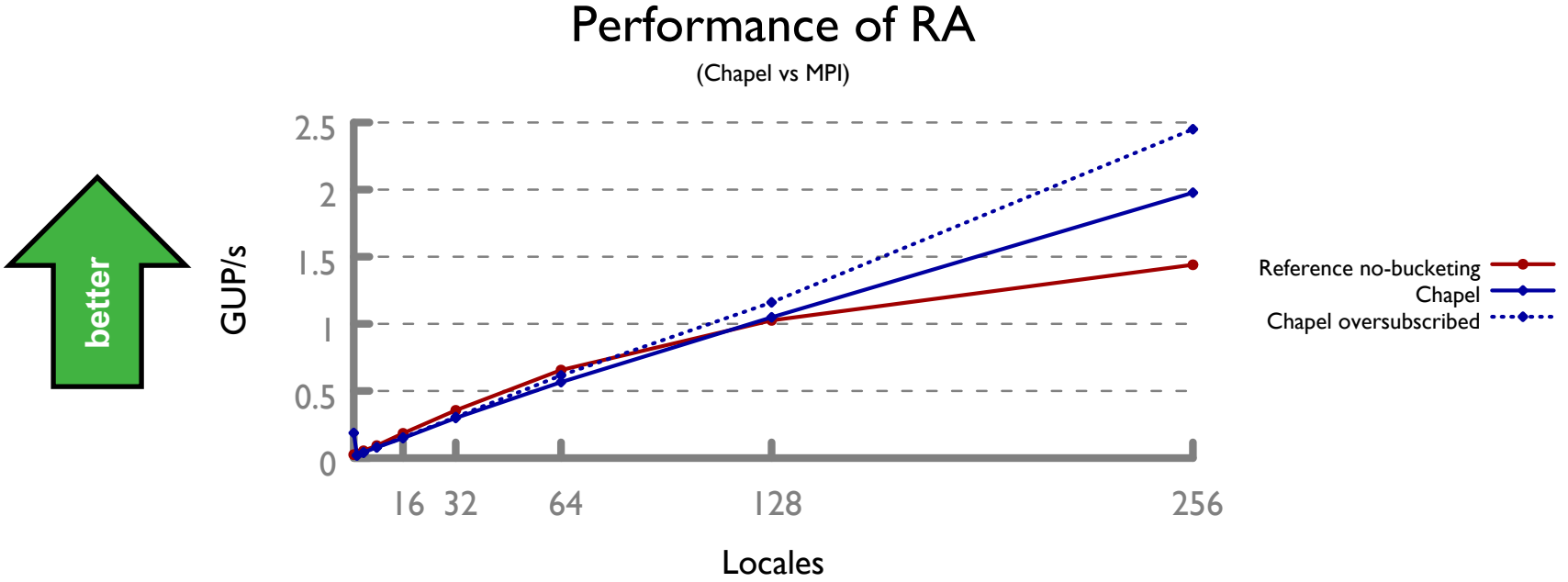
```
UpdateDist = new dmap(new Block(boundingBox={0..N_U-1}));
```



HPCC RA Performance: Chapel vs. MPI

Update Dist Declaration

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



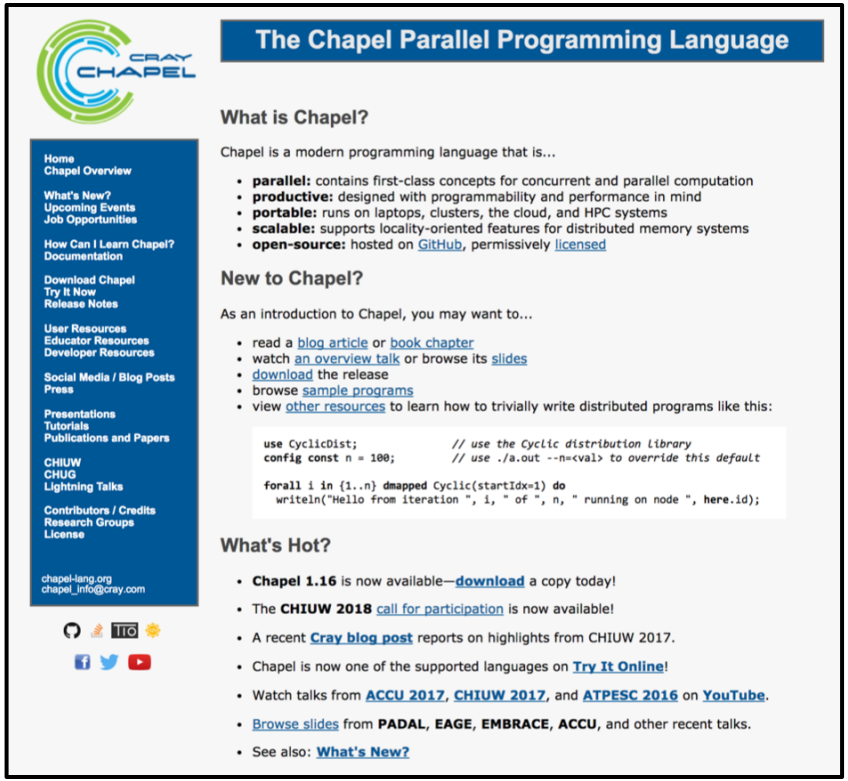
Wrapping Up

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/



How to Track Chapel

http://facebook.com/ChapelLanguage

http://twitter.com/ChapelLanguage

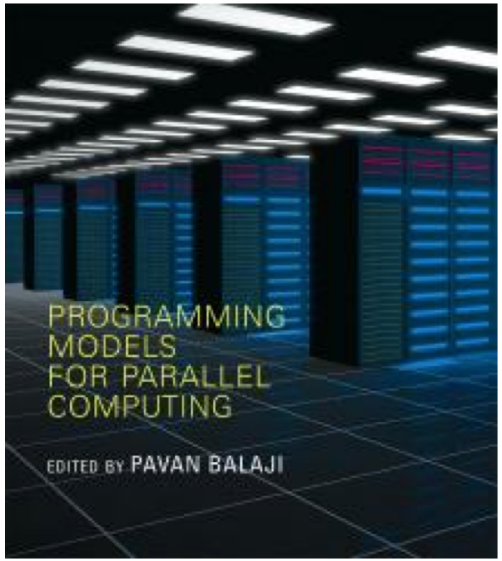
https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/

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**

Suggested Reading (short attention spans)

CHIUW 2017: Surveying the Chapel Landscape, Cray Blog, July 2017.

- *a run-down of recent events*

Chapel: Productive Parallel Programming, Cray Blog, May 2013.

- *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*

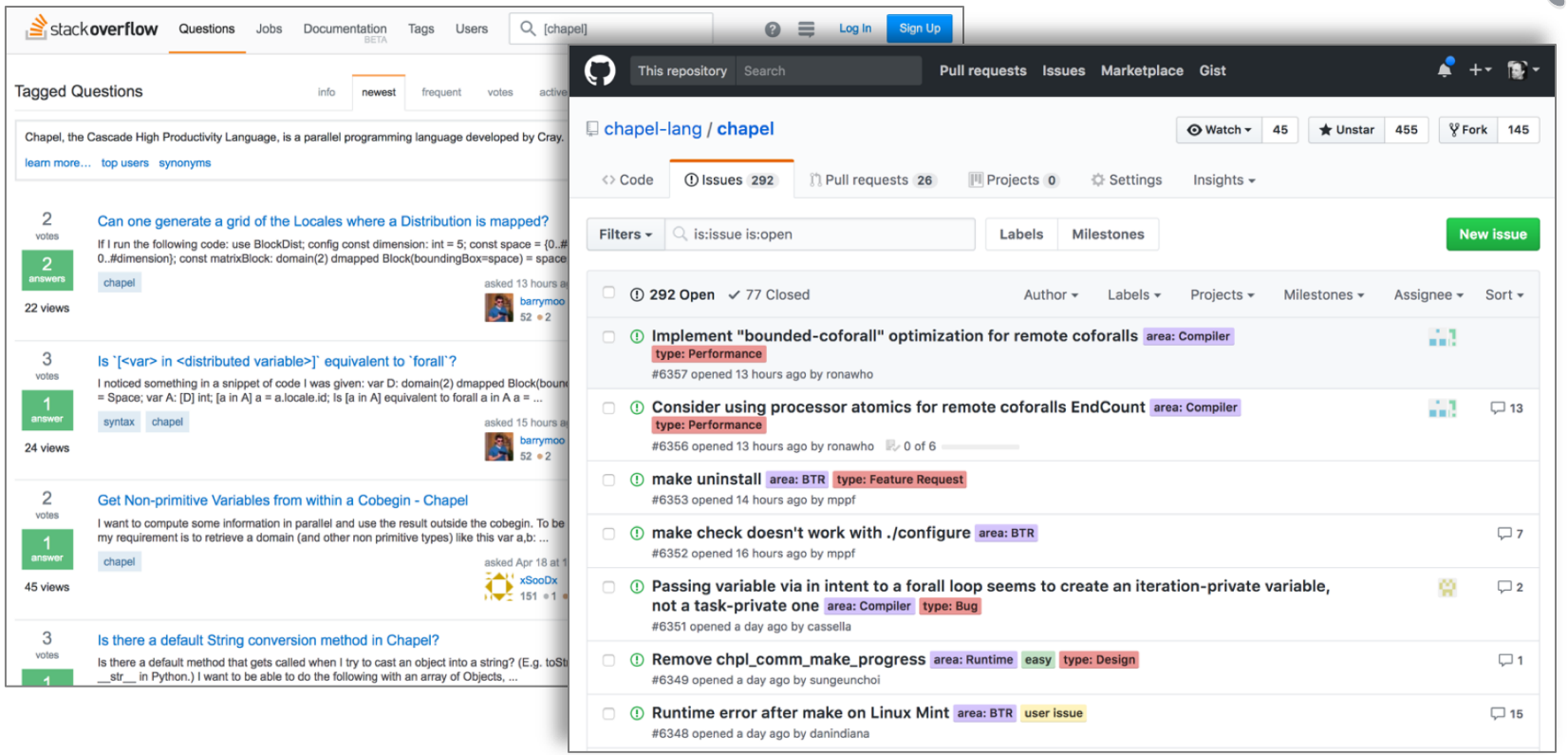
Why Chapel? **(parts 1, 2, 3)**, Cray Blog, Jun-Oct 2014.

- *a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges*

[Ten] Myths About Scalable Programming Languages, IEEE TCSC Blog (index available on chapel-lang.org "blog posts" page), Apr-Nov 2012.

- *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.

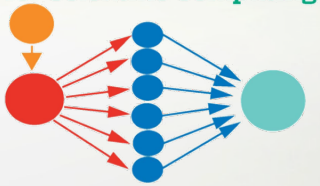
Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

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.

SIAM Conference on
Parallel Processing
for Scientific Computing



CRAY
THE SUPERCOMPUTER COMPANY