Chapel Comes of Age: Productive Parallelism at Scale

Brad Chamberlain, Chapel Team, Cray Inc.

PNW PLSE Workshop

May 14, 2018

Chapel:
Niche or Quiche?

Brad Chamberlain, Chapel Team, Cray Inc.

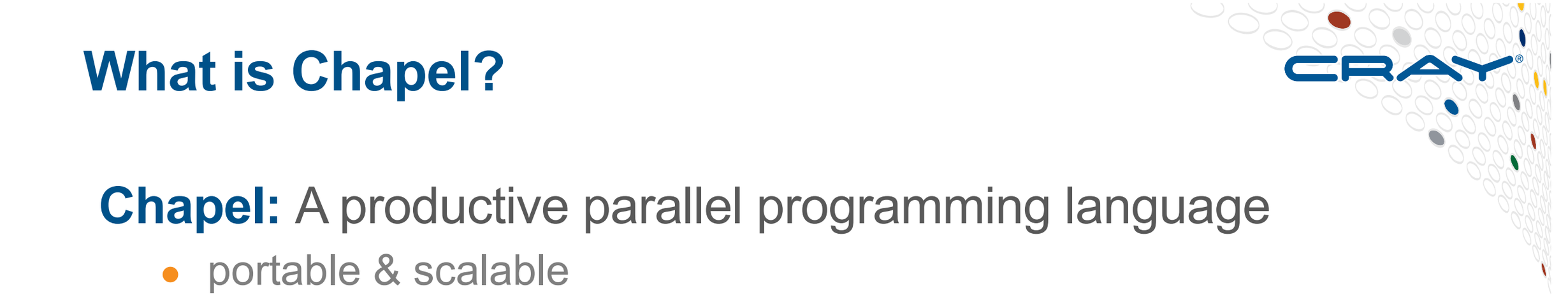
PNW PLSE Workshop

May 14, 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?



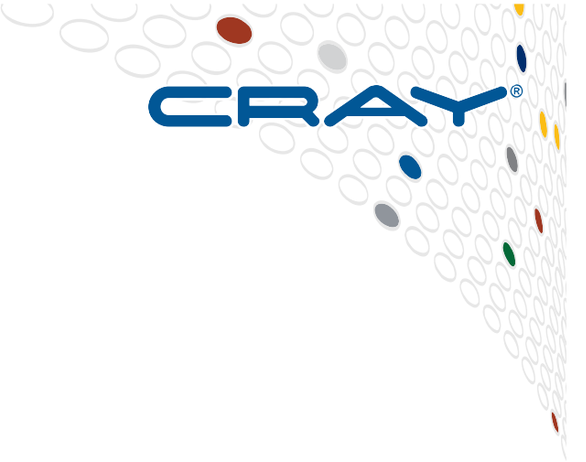
Chapel: A productive parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

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

Chapel and Productivity



Chapel aims to be as…

…**programmable** as Python

…**fast** as Fortran

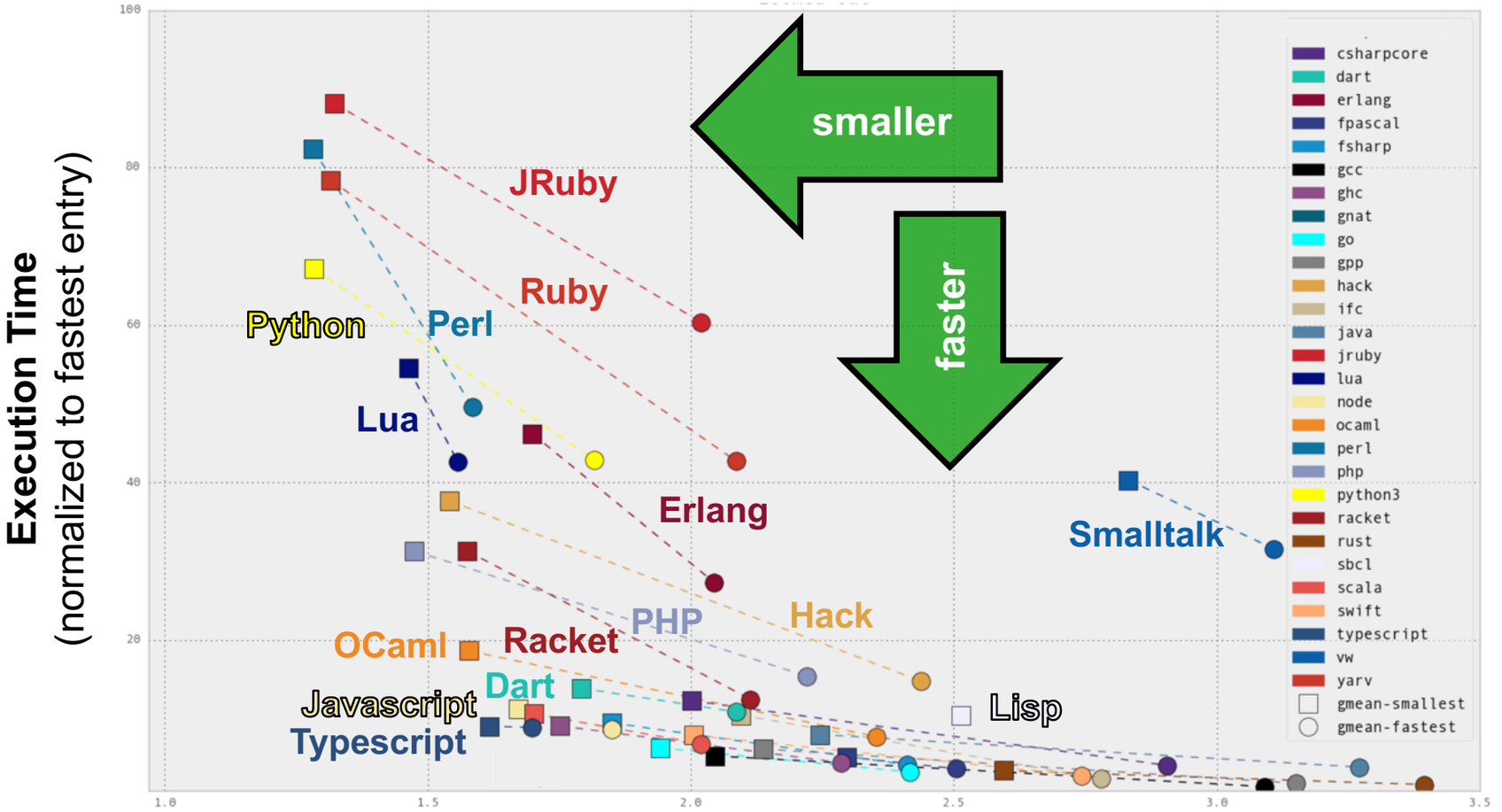
…**scalable** as MPI

…**portable** as C

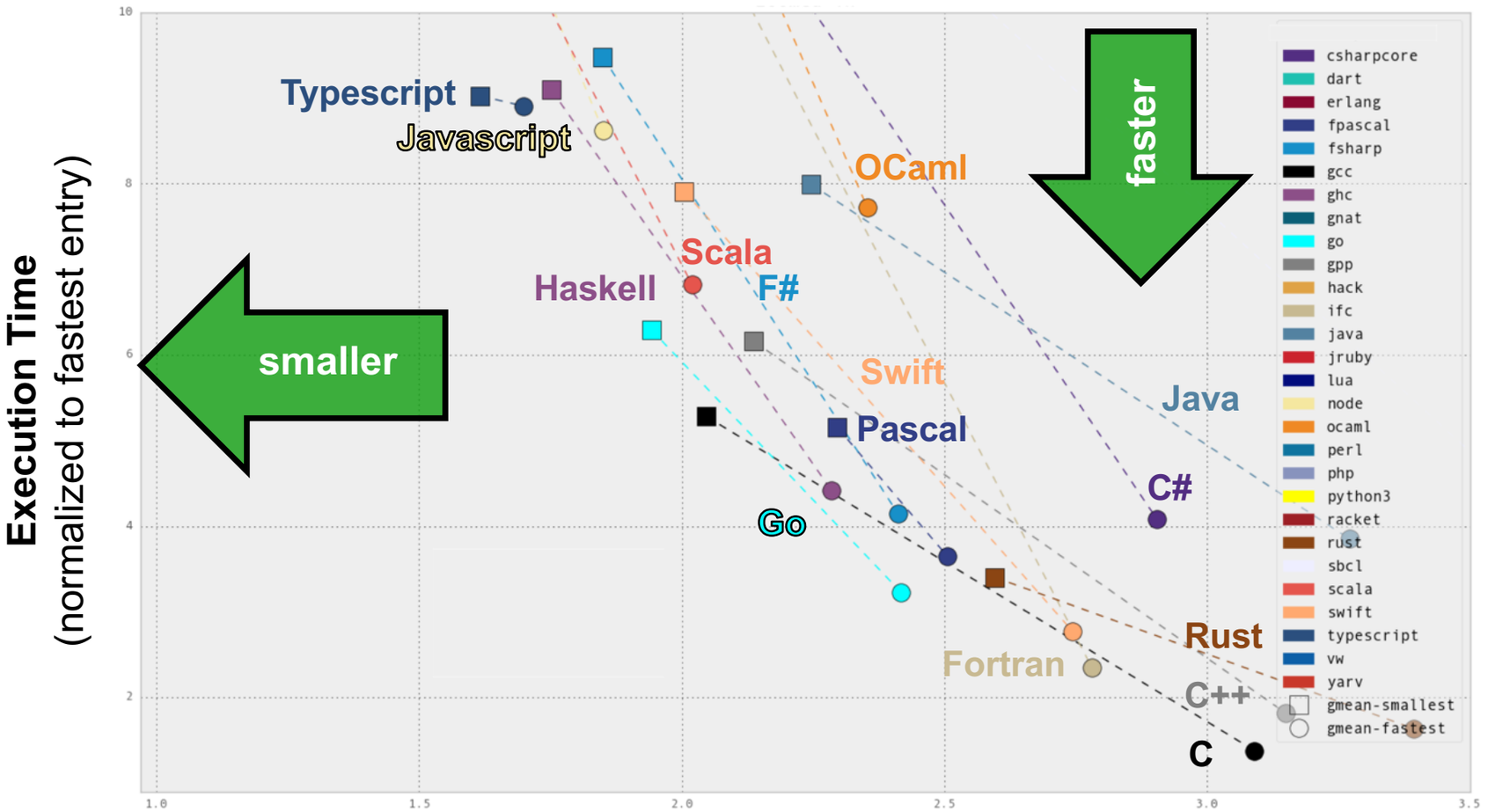
…**flexible** as C++

…**fun** as [your favorite programming language]

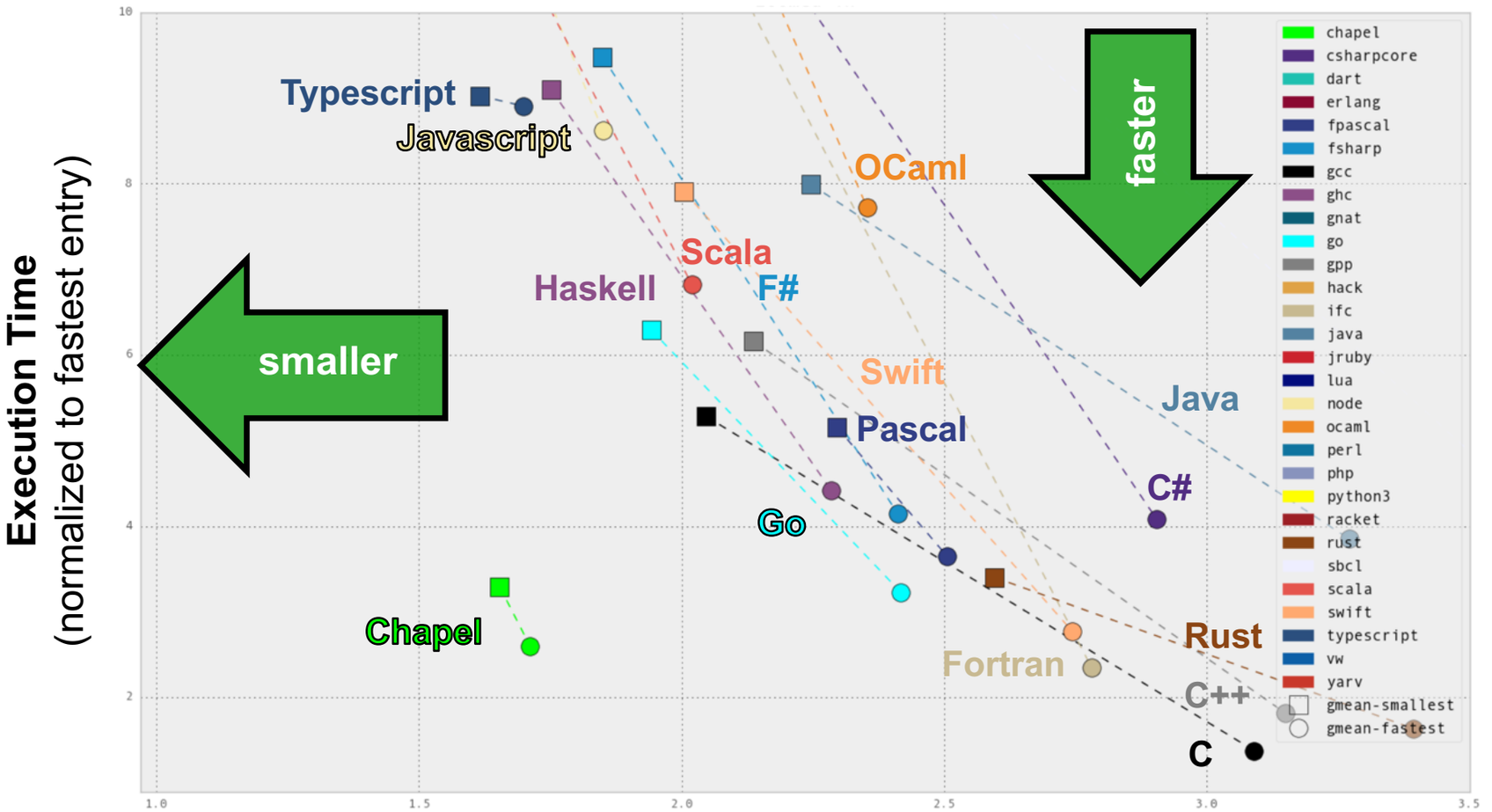
CLBG Cross-Language Summary
(Oct 2017 standings)



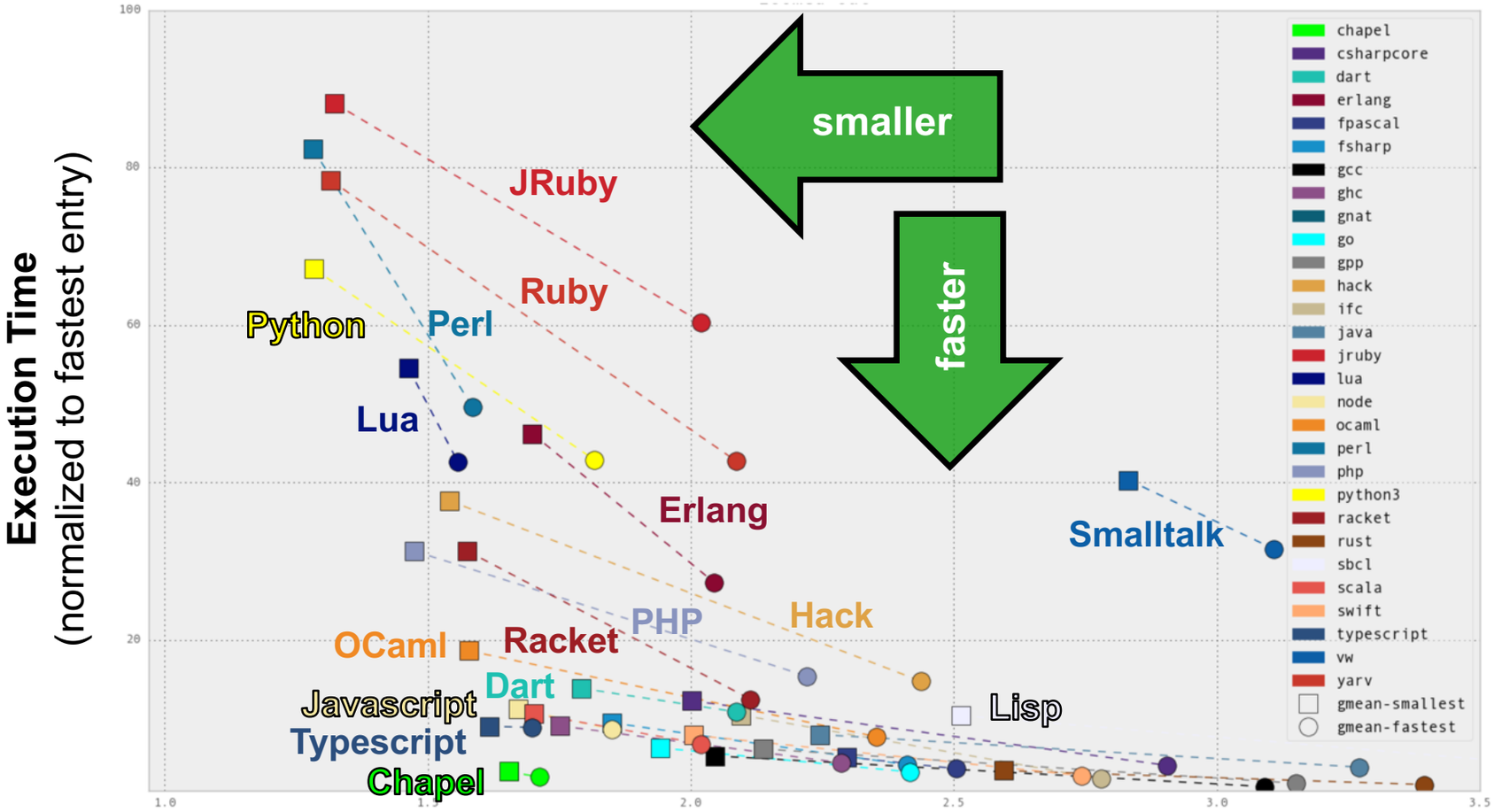
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)



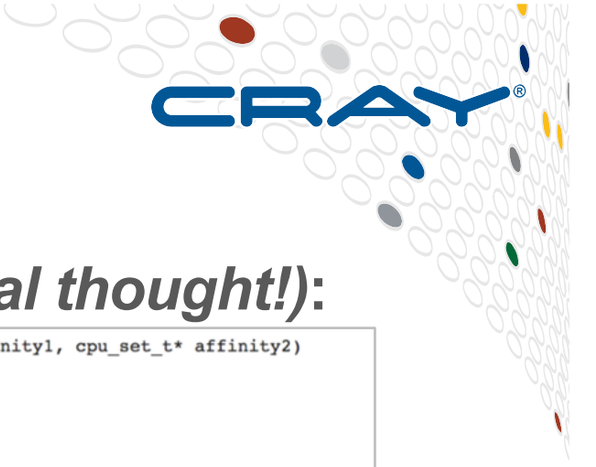
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)



CLBG Cross-Language Summary
(Oct 2017 standings)



CLBG: Qualitative Code Comparisons



Can also browse program source code *(but this requires actual thought!)*:

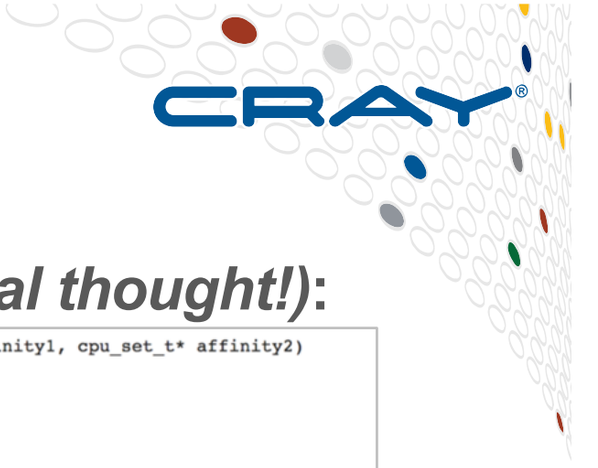
```
proc main() {
  printColorEquations();

  const group1 = [i in 1..popSize1] new Chameneos(i, ((i-1)%3):Color);
  const group2 = [i in 1..popSize2] new Chameneos(i, colors10[i]);

  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }

  print(group1);
  print(group2);

  for c in group1 do delete c;
  for c in group2 do delete c;
}


//
// Print the results of getNewColor() for all color pairs.
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " -> ", getNewColor(c1, c2));
  writeln();
}


//
// Hold meetings among the population by creating a shared meeting
// place, and then creating per-chameneos tasks to have meetings.
//
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do           // create a task per chameneos
    c.haveMeetings(place, population);

  delete place;
}
```

excerpt from 1210 gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t                   active_cpus;
    FILE*                       f;
    char                        buf [2048];
    char const*                 pos;
    int                         cpu_idx;
    int                         physical_id;
    int                         core_id;
    int                         cpu_cores;
    int                         apic_id;
    size_t                      cpu_count;
    size_t                      i;

    char const*                 processor_str       = "processor";
    size_t                      processor_str_len   = strlen(processor_str);
    char const*                 physical_id_str     = "physical id";
    size_t                      physical_id_str_len = strlen(physical_id_str);
    char const*                 core_id_str         = "core id";
    size_t                      core_id_str_len     = strlen(core_id_str);
    char const*                 cpu_cores_str       = "cpu cores";
    size_t                      cpu_cores_str_len   = strlen(cpu_cores_str);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }

    if (cpu_count == 1)
    {
        is_smp[0] = 0;
        return;
    }

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 2863 gz C gcc entry

CLBG: Qualitative Code Comparisons



Can also browse program source code *(but this requires actual thought!)*:

```
proc main() {
  printColorEquations();

  const group1 = [i in 1..popSize1] new Chameneos(i, (
  const group2 = [i in 1..popSize2] new Chameneos(i, c

  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }

  print(group1);
  print(group2);

  for c in group1 do delete c;
  for c in group2 do delete c;
}


//
// Print the results of getNewColor() for all color pa
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " -> ", getNewColor(c1, c
  writeln();
}


//
// Hold meetings among the population by creating a sh
// place, and then creating per-chameneos tasks to hav
//
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do           // create a ta
    c.haveMeetings(place, population);

  delete place;
}
```

```
cobegin {
  holdMeetings(group1, n);
  holdMeetings(group2, n);
}
```

```
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do              // creat
    c.haveMeetings(place, population);

  delete place;
}
```

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)

    active_cpus;
    f;
    buf [2048];
    pos;
    cpu_idx;
    physical_id;
    core_id;
    cpu_cores;
    apic_id;
    cpu_count;
    i;

    processor_str       = "processor";
    processor_str_len   = strlen(processor_str);
    physical_id_str     = "physical id";
    physical_id_str_len = strlen(physical_id_str);
    core_id_str         = "core id";
size_t                  n(core_id_str);
char const*             cores";
size_t                  n(cpu_cores_str);
char const*

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 1210 gz Chapel entry

excerpt from 2863 gz C gcc entry

CLBG: Qualitative Code Comparisons

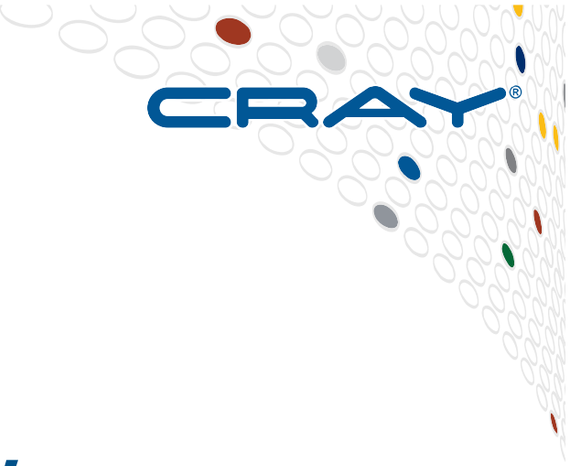
Can also browse program source code *(but this requires actual thought!)*:

```
proc main() {
char const*                 core_id_str          = "core id"
size_t                      core_id_str_len      = strlen(co
char const*                 cpu_cores_str        = "cpu core
size_t                      cpu_cores_str_len    = strlen(cp

CPU_ZERO(&active_cpus);
sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
cpu_count = 0;
for (i = 0; i != CPU_SETSIZE; i += 1)
{
    if (CPU_ISSET(i, &active_cpus))
    {
        cpu_count += 1;
    }
}

if (cpu_count == 1)
{
    is_smp[0] = 0;
    return;
}
```

excerpt from 1210 gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t                   active_cpus;
    FILE*                       f;
    char                        buf [2048];
    char const*                 pos;
    int                         cpu_idx;
    int                         physical_id;
    int                         core_id;
    int                         cpu_cores;
    int                         apic_id;
    size_t                      cpu_count;
    size_t                      i;

    char const*                 processor_str       = "processor";
    size_t                      processor_str_len   = strlen(processor_str);
    char const*                 physical_id_str     = "physical id";
    size_t                      physical_id_str_len = strlen(physical_id_str);
    char const*                 core_id_str         = "core id";
    size_t                      core_id_str_len     = strlen(core_id_str);
    char const*                 cpu_cores_str       = "cpu cores";
    size_t                      cpu_cores_str_len   = strlen(cpu_cores_str);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }

    if (cpu_count == 1)
    {
        is_smp[0] = 0;
        return;
    }

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

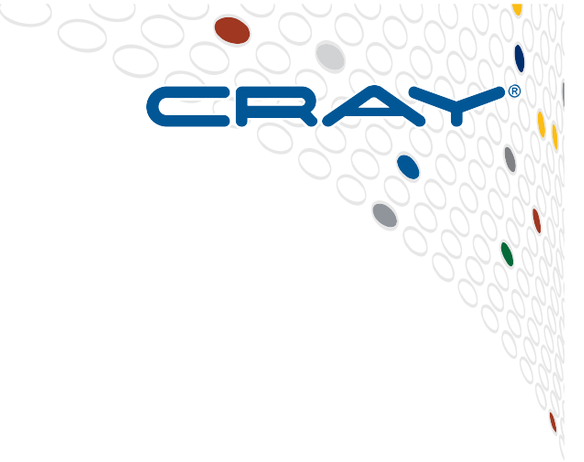
excerpt from 2863 gz C gcc entry



Excerpt from PNW PLSE Review

"Chapel has been around for quite a while, and it still seems like a niche language…"

Chapel: "A Niche Language"?



Chapel is arguably niche in that it…

…was originally designed for HPC

Yet, Chapel's chief concerns aren't HPC-specific:

- performance
- programmability (cf. Python)
- parallelism (cf. multicore)
- distributed memory (cf. cloud computing)

Chapel: "A Niche Language"?

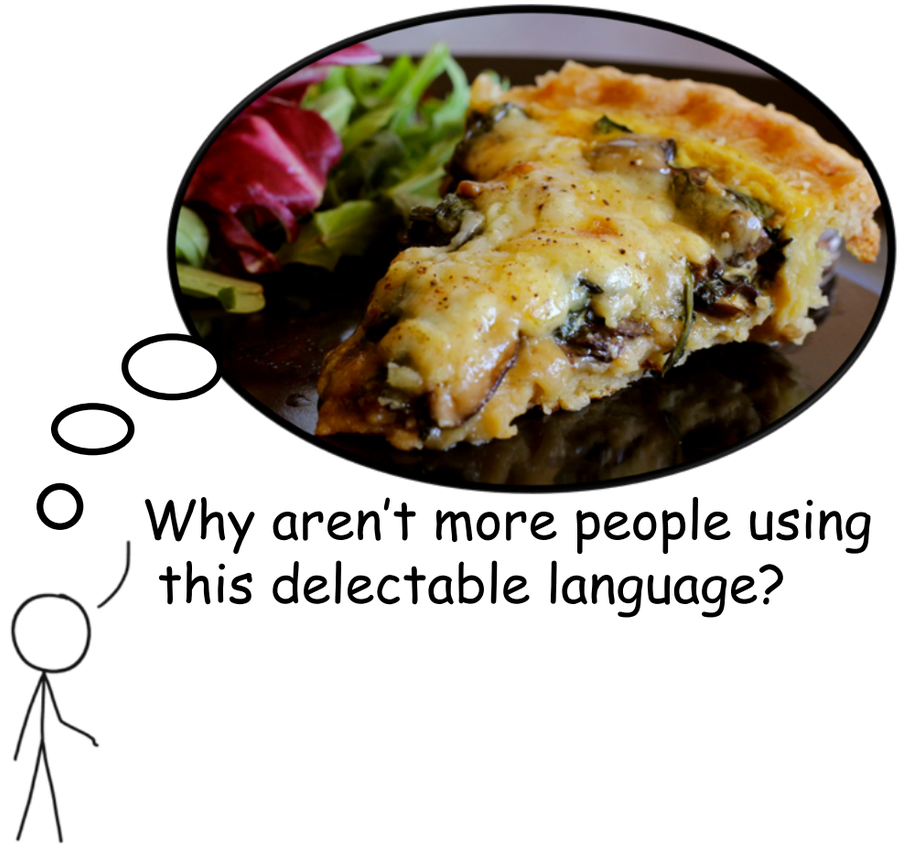
Chapel is arguably niche in that it…

- …was originally designed for HPC
- …has only a modest-sized community (so far)

Yet, we've historically discouraged its use in production…

Chapel: A Quiche Language!

The outsider's impression:

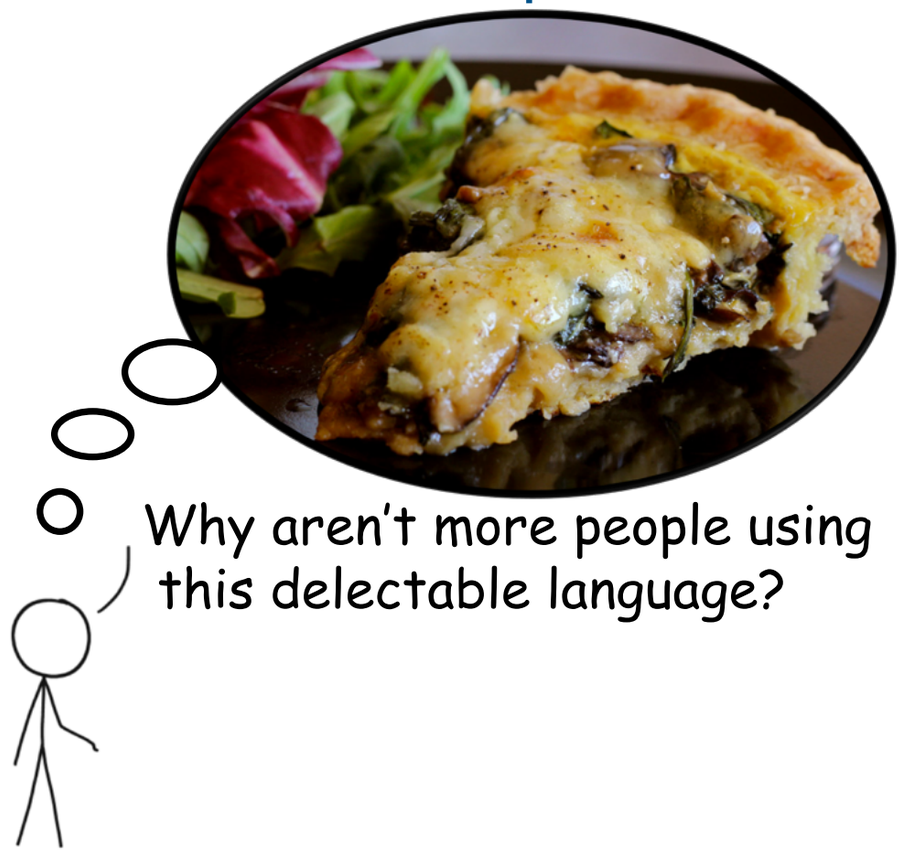


The reality, for most of Chapel's history:

Image sources: https://www.themountainkitchen.com, https://xkcd.com/

Chapel: A Quiche Language!

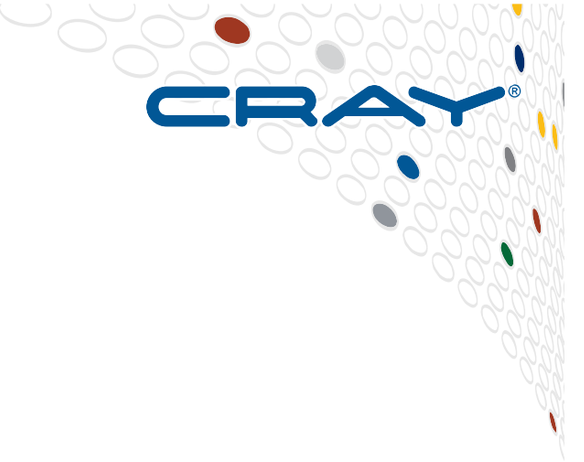
The outsider's impression:



Though recently, it's more like:

Image sources: https://www.themountainkitchen.com, https://xkcd.com/

Chapel: "Been Around for Quite Awhile"



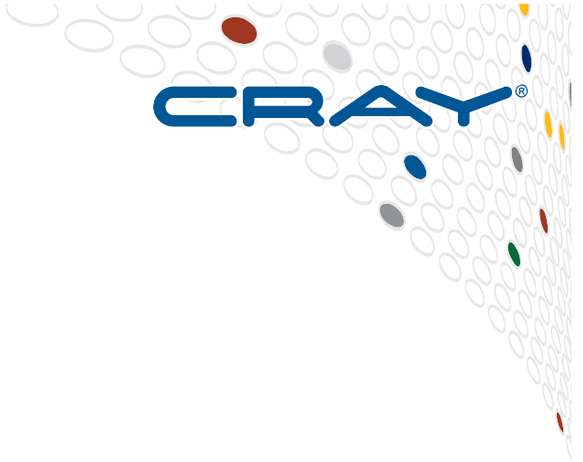
Chapel's Infancy: DARPA HPCS (2003–2012)

- Research focus: ~6-7 FTEs
  - distinguish locality from parallelism
  - seamlessly mix data- and task-parallelism
  - support user-defined distributed arrays, parallel iterators

Chapel's Adolescence: "the five-year push" (2013–2018)

- Development focus: ~13-14 FTEs
  - **performance and scalability**
  - **ecosystem:** documentation, libraries, tools, …
  - **base language fixes:** OOP features, error-handling, strings, …

Chapel: "Been Around for Quite Awhile"



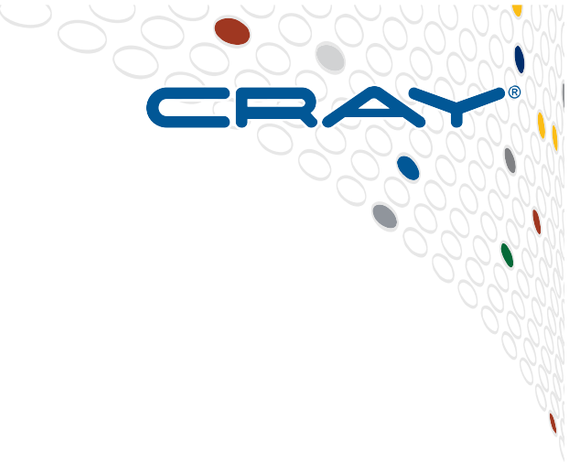
Chapel's Infancy: DARPA HPCS (2003–2012)

- Research focus: ~6-7 FTEs
  - distinguish locality from parallelism
  - seamlessly mix data- and task-parallelism
  - support user-defined distributed arrays, parallel iterators

Chapel's Adolescence: "the five-year push" (2013–2018)

Then **Now**

- Development focus: ~13-14 FTEs
  - **performance and scalability**
  - **ecosystem:** documentation, libraries, tools, …
  - **base language fixes:** OOP features, error-handling, strings, …

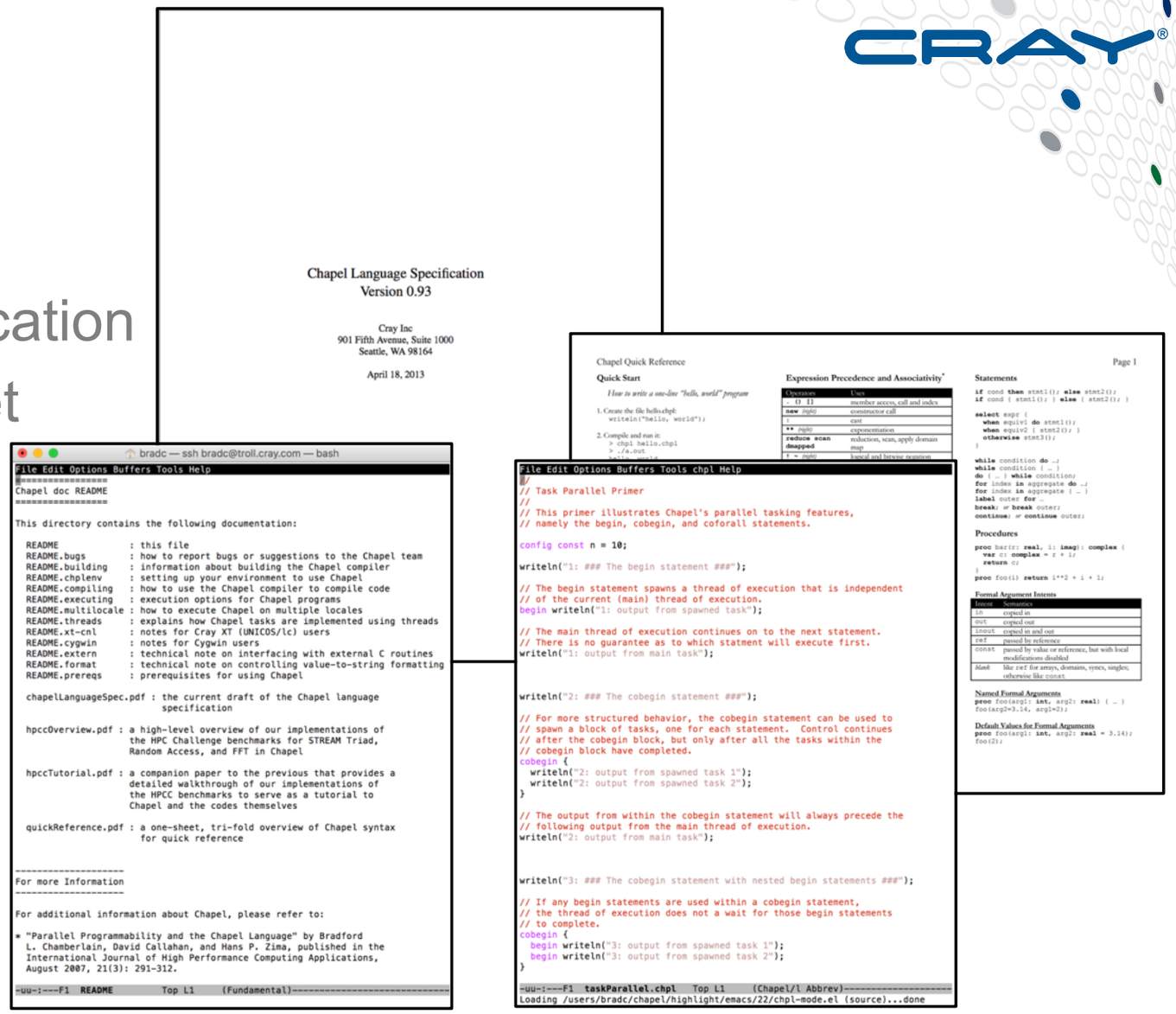


Chapel Ecosystem: Then vs. Now

Documentation: Then

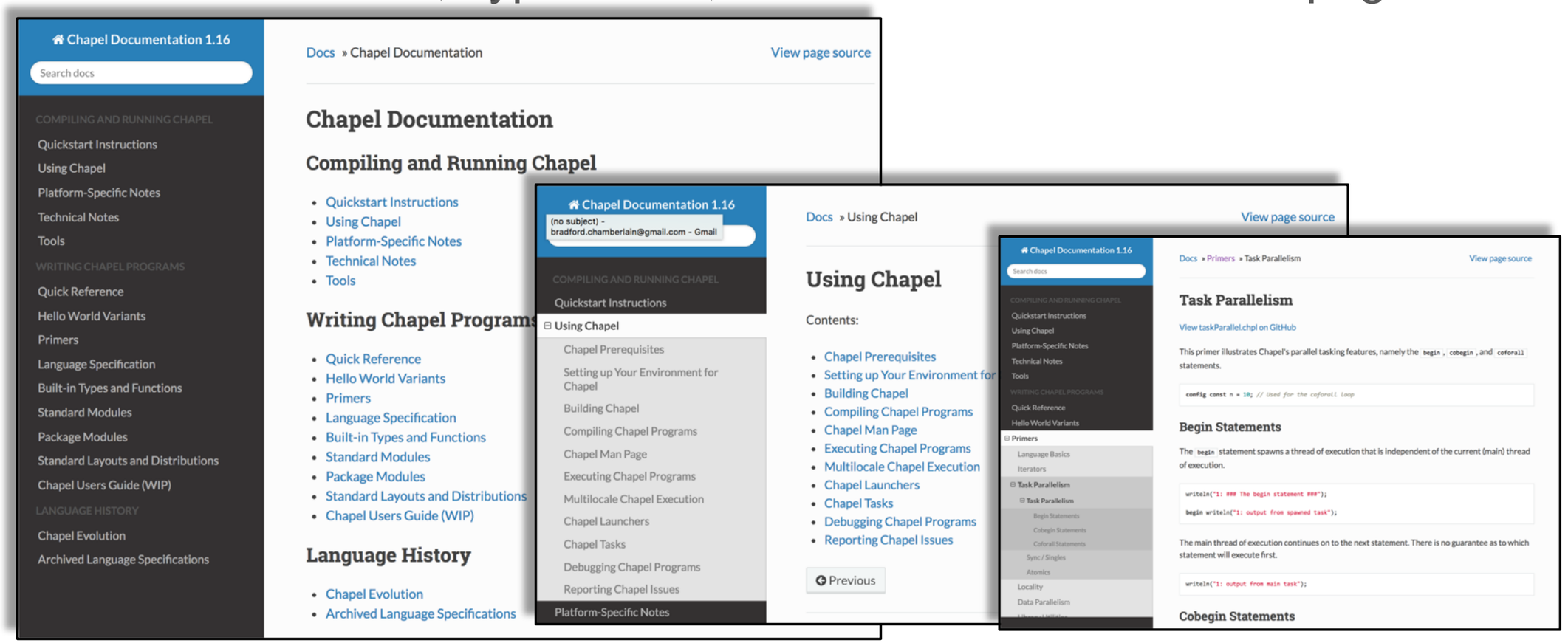
After HPCS:

- a PDF language specification
- a Quick Reference sheet
- a number of READMEs
- ~22 primer examples



Documentation: Now

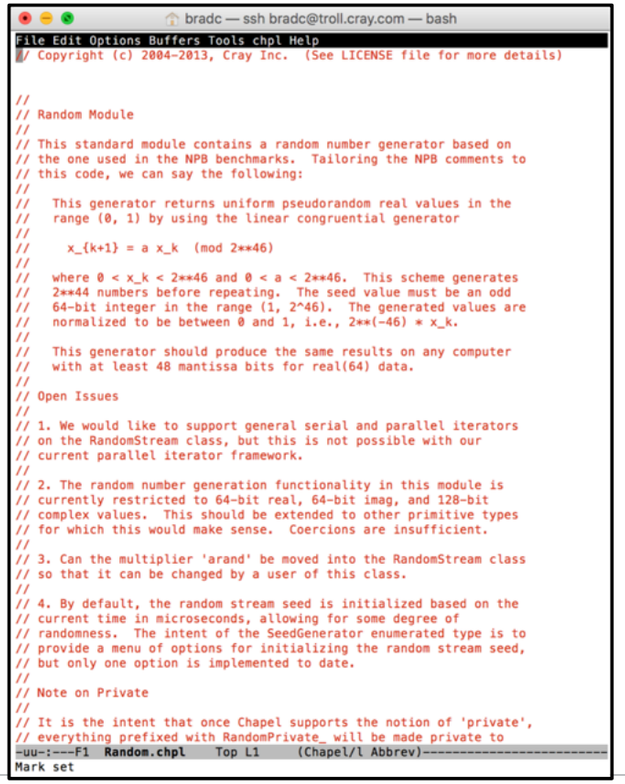
Now: 200+ modern, hyperlinked, web-based documentation pages

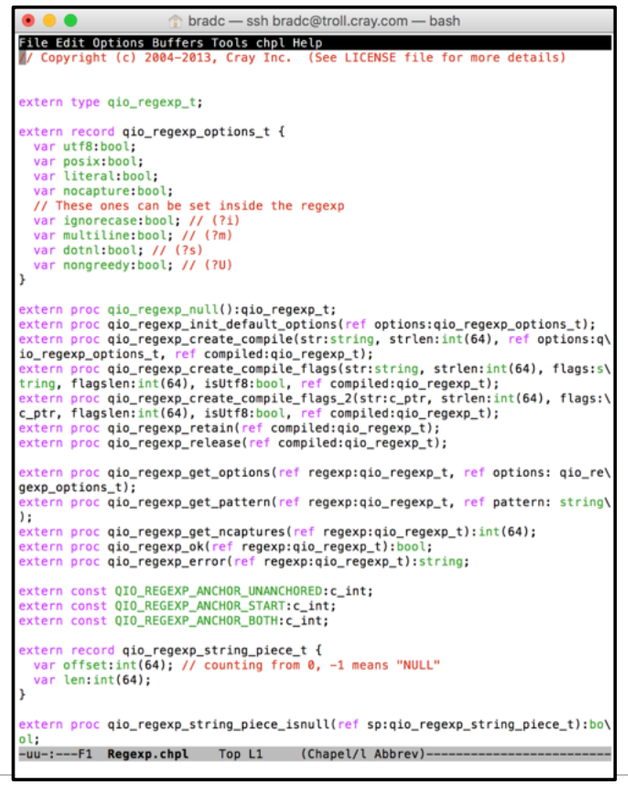


Libraries: Then

After HPCS: ~25 library modules

- documented via source comments, if at all:

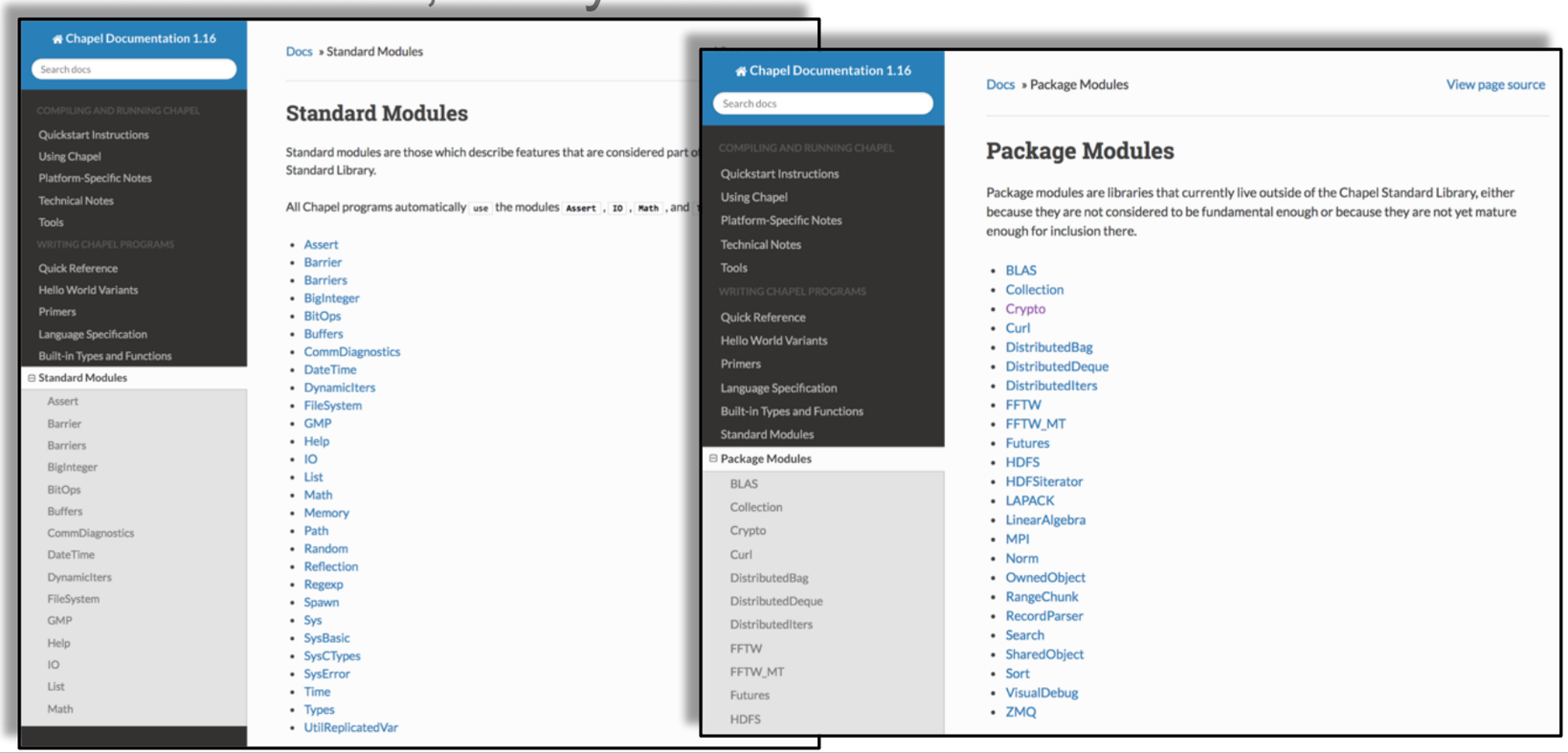




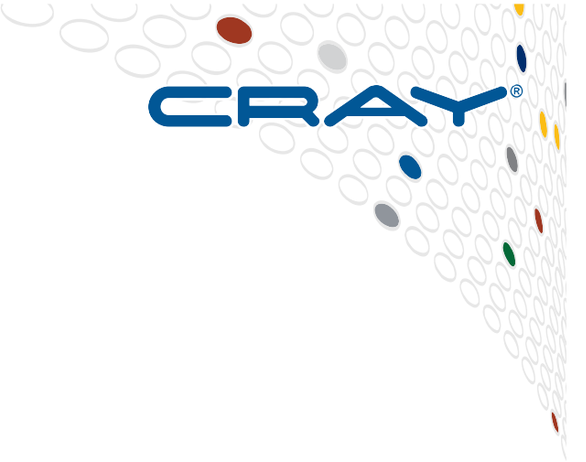
Libraries: Now

Now: ~60 library modules

- web-documented, many user-contributed



Tools: Then



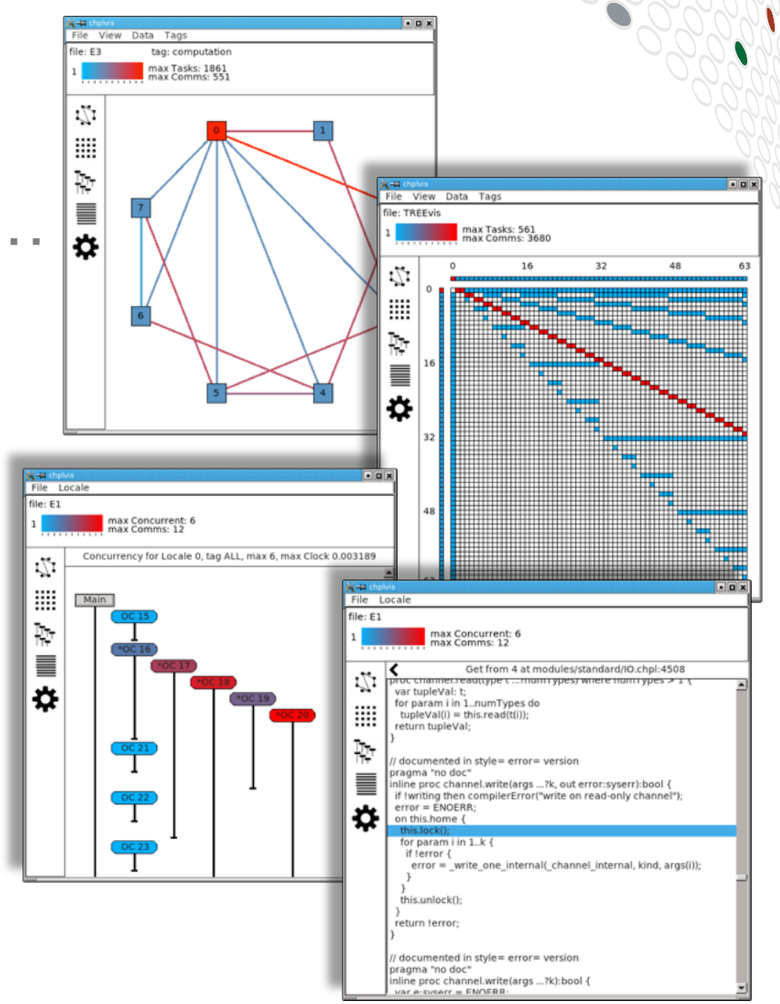
After HPCS:

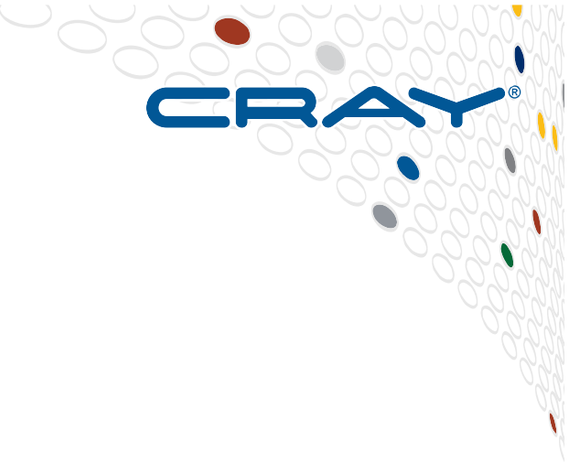
- **highlighting modes** for emacs and vim
- **chpldoc:** documentation tool (rough draft)

Tools: Now

Now:

- **highlighting modes** for emacs, vim, atom, …
- **chpldoc:** documentation tool
- **mason:** package manager
- **c2chapel:** interoperability aid
- **chpltags:** helps search Chapel code
- **bash tab completion:** command-line help
- **chplvis:** performance visualizer / debugger



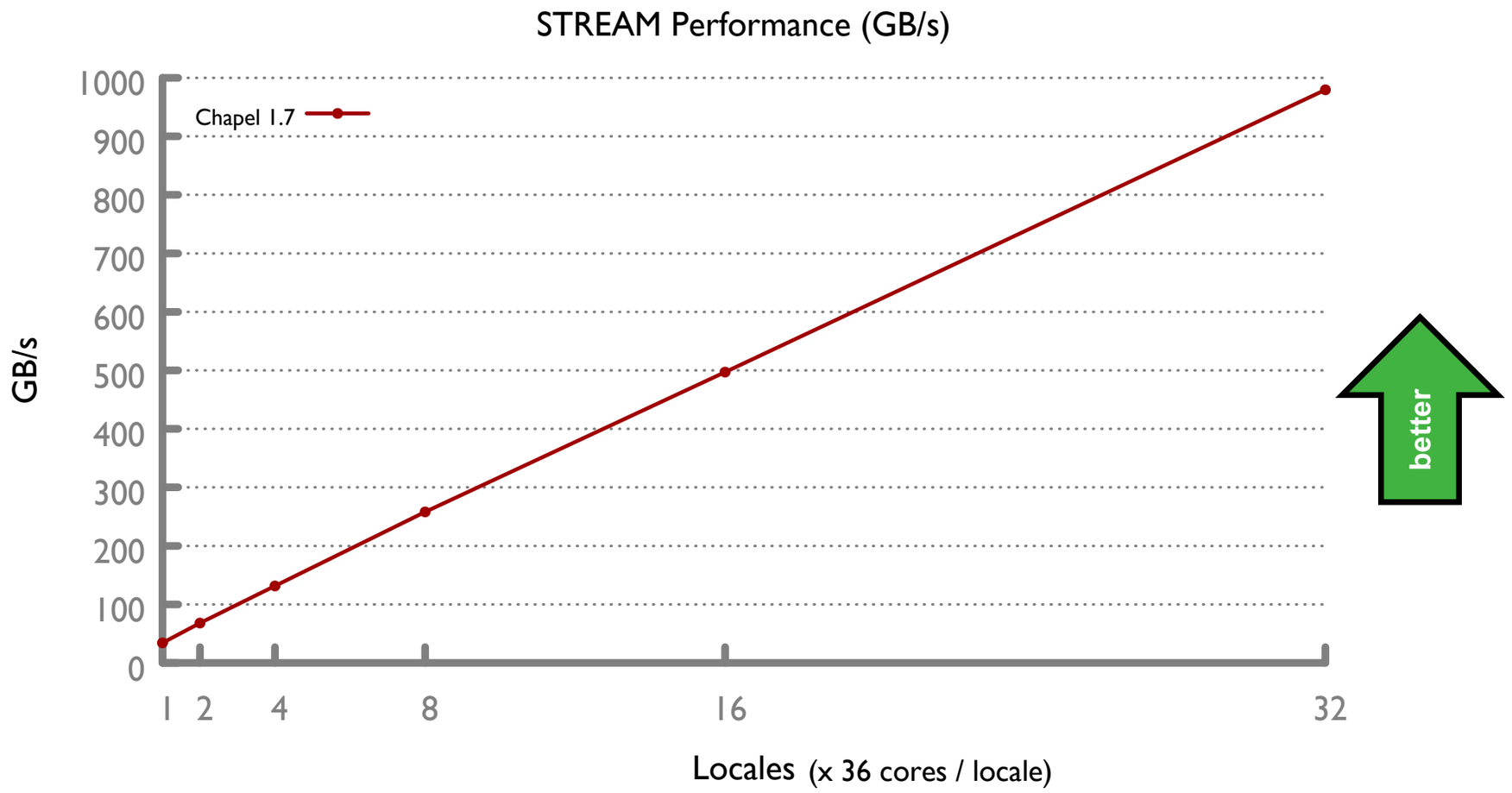


Chapel Performance: Then vs. Now

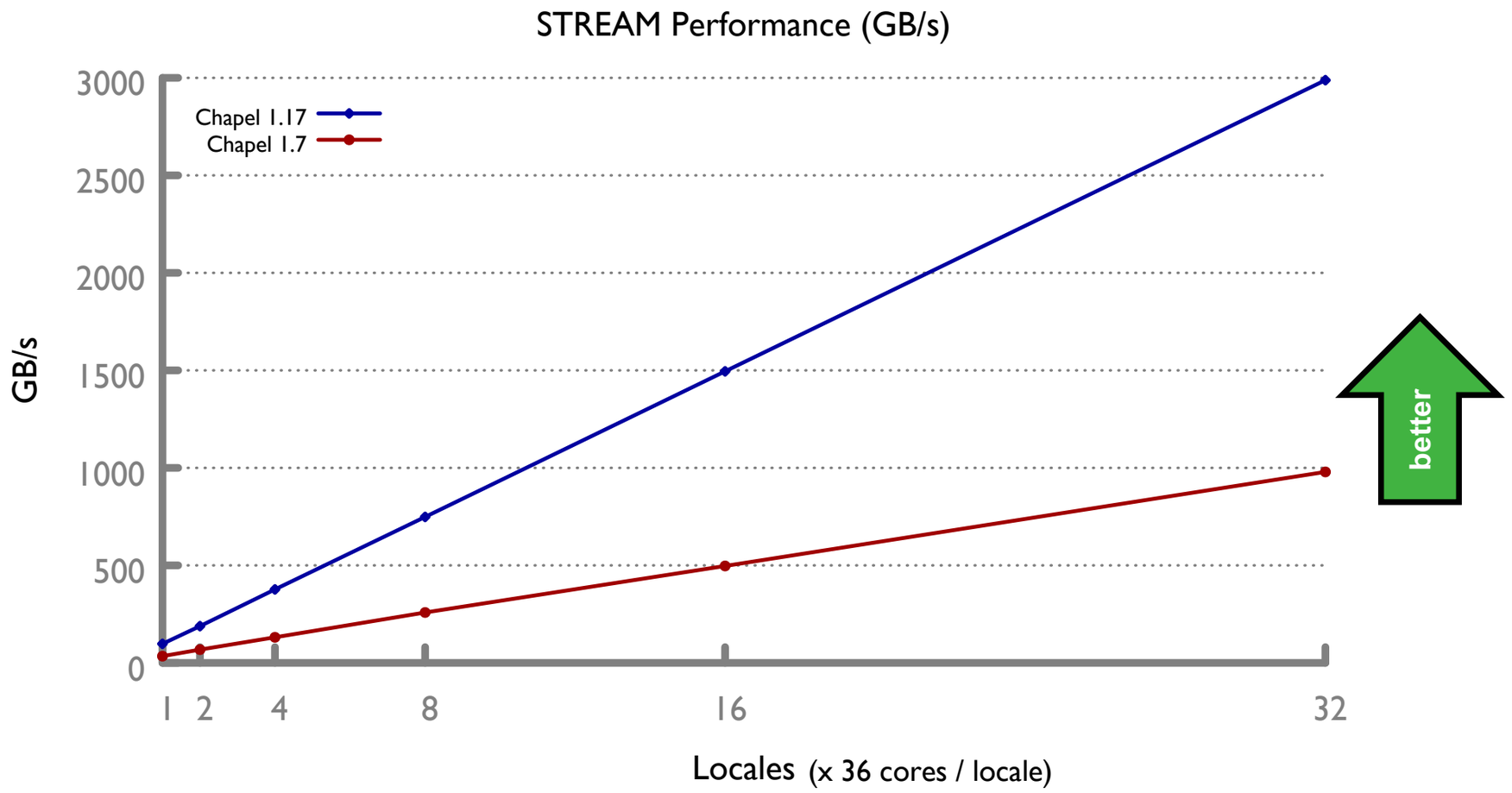
Performance Focus Areas during 5-year push

- Cleaner, simpler generated code
- NUMA sensitivity within multi-socket nodes
- Best-use of RDMA and NIC memory registration
- Reduced overheads in tasks, memory, communication
- Bulk transfer optimizations
- …and much more…

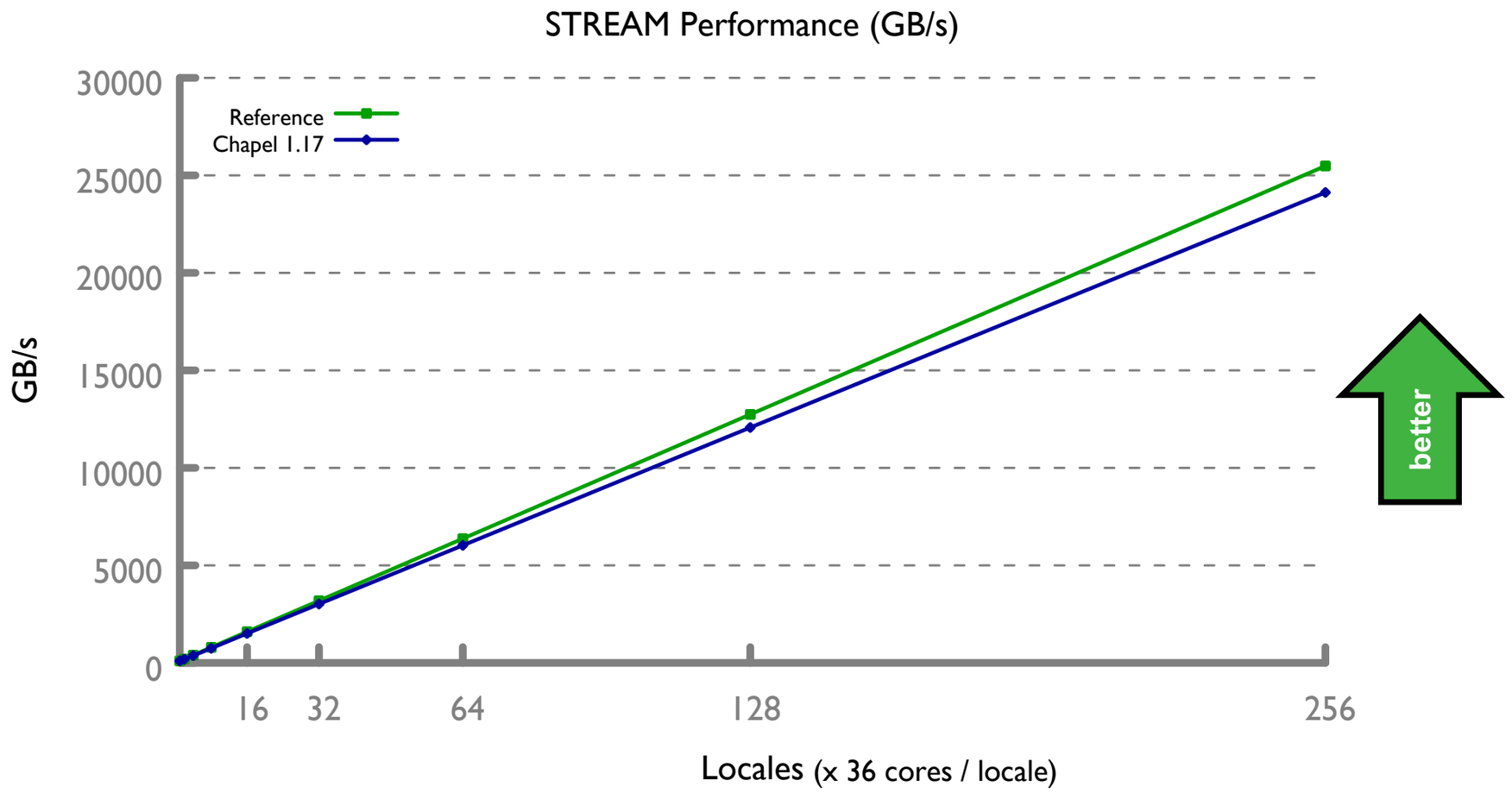
STREAM Triad Performance: Chapel Then



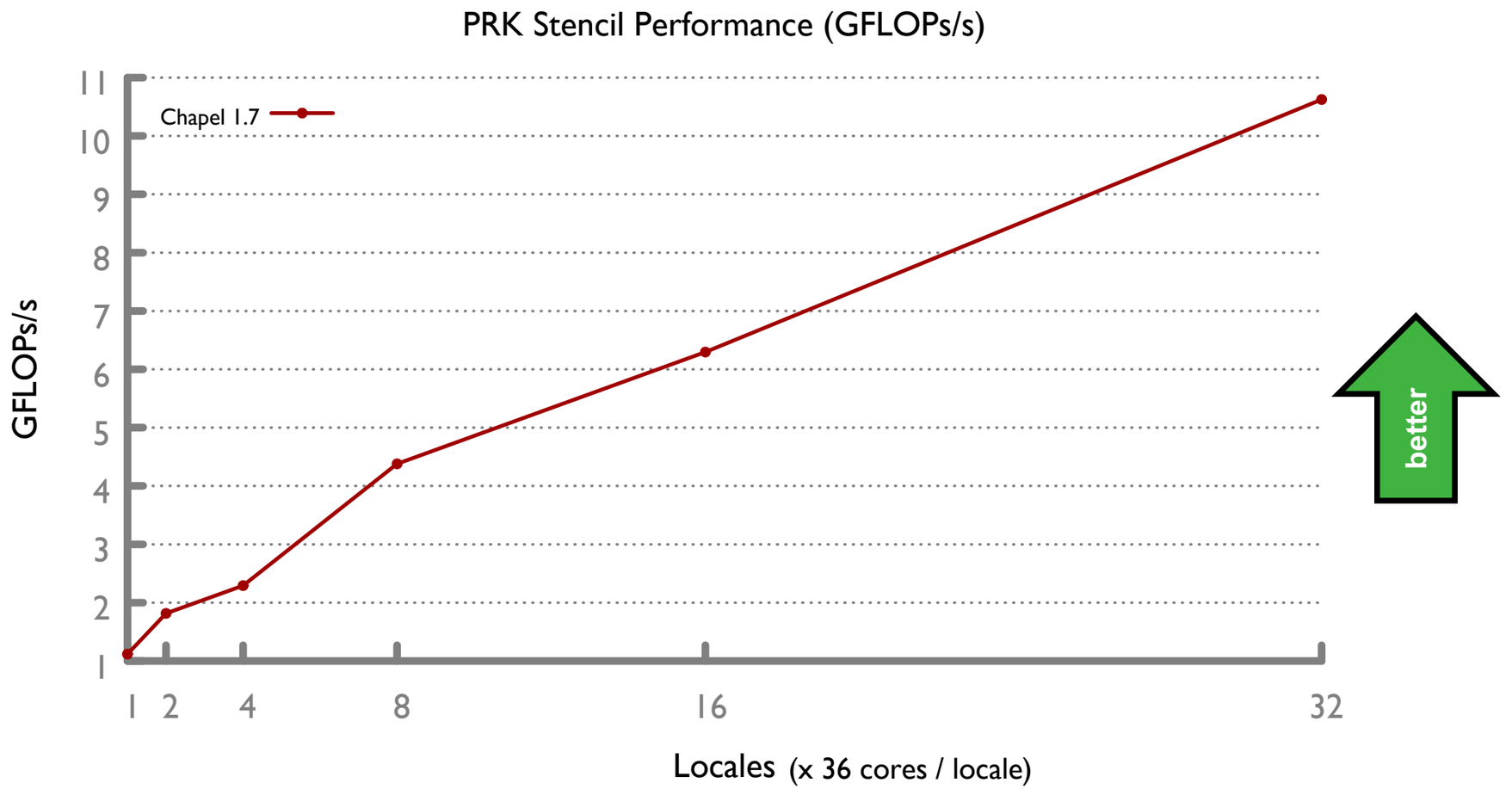
STREAM Triad Performance: Chapel Then vs. Now



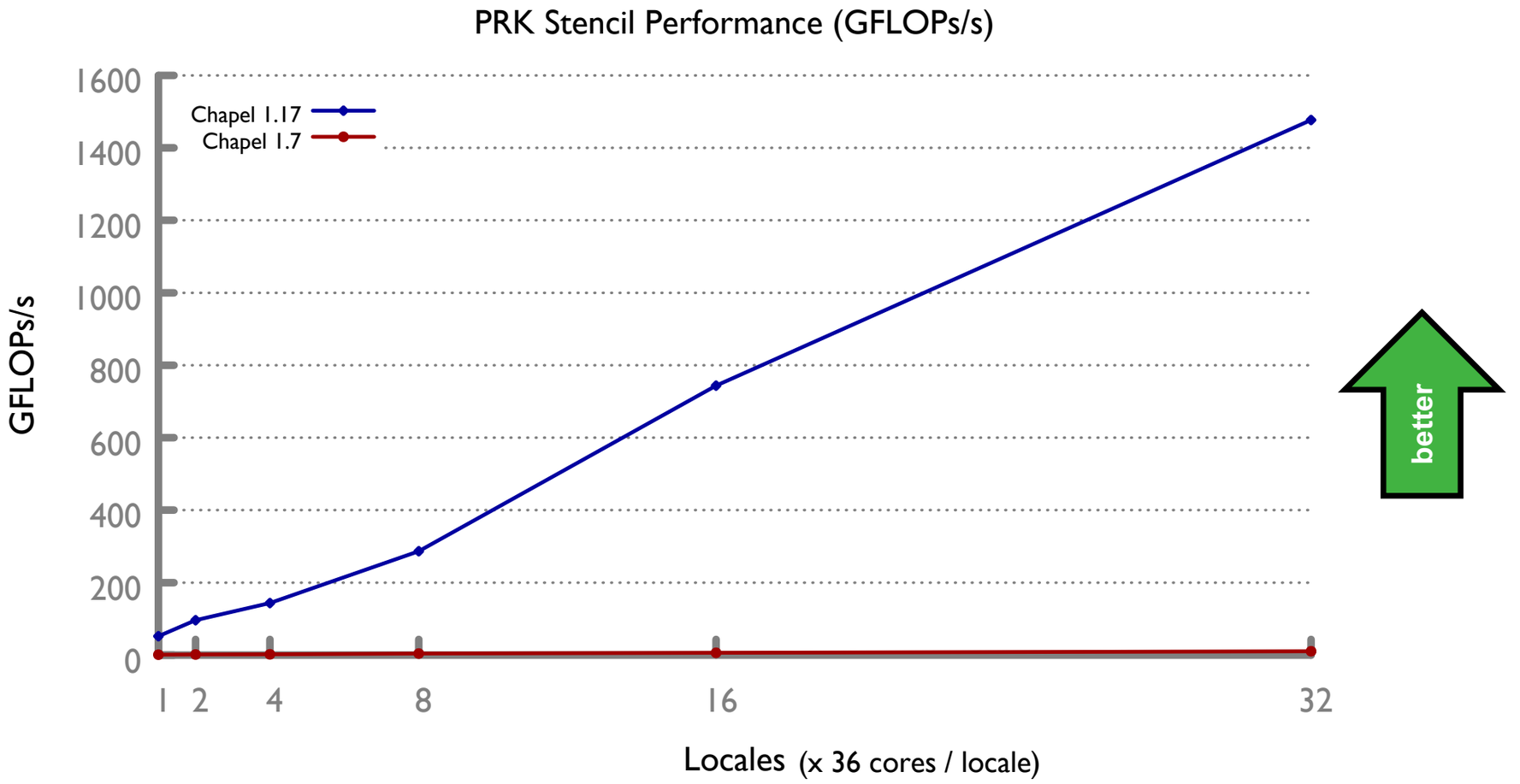
STREAM Triad Performance: Chapel Now vs. ref



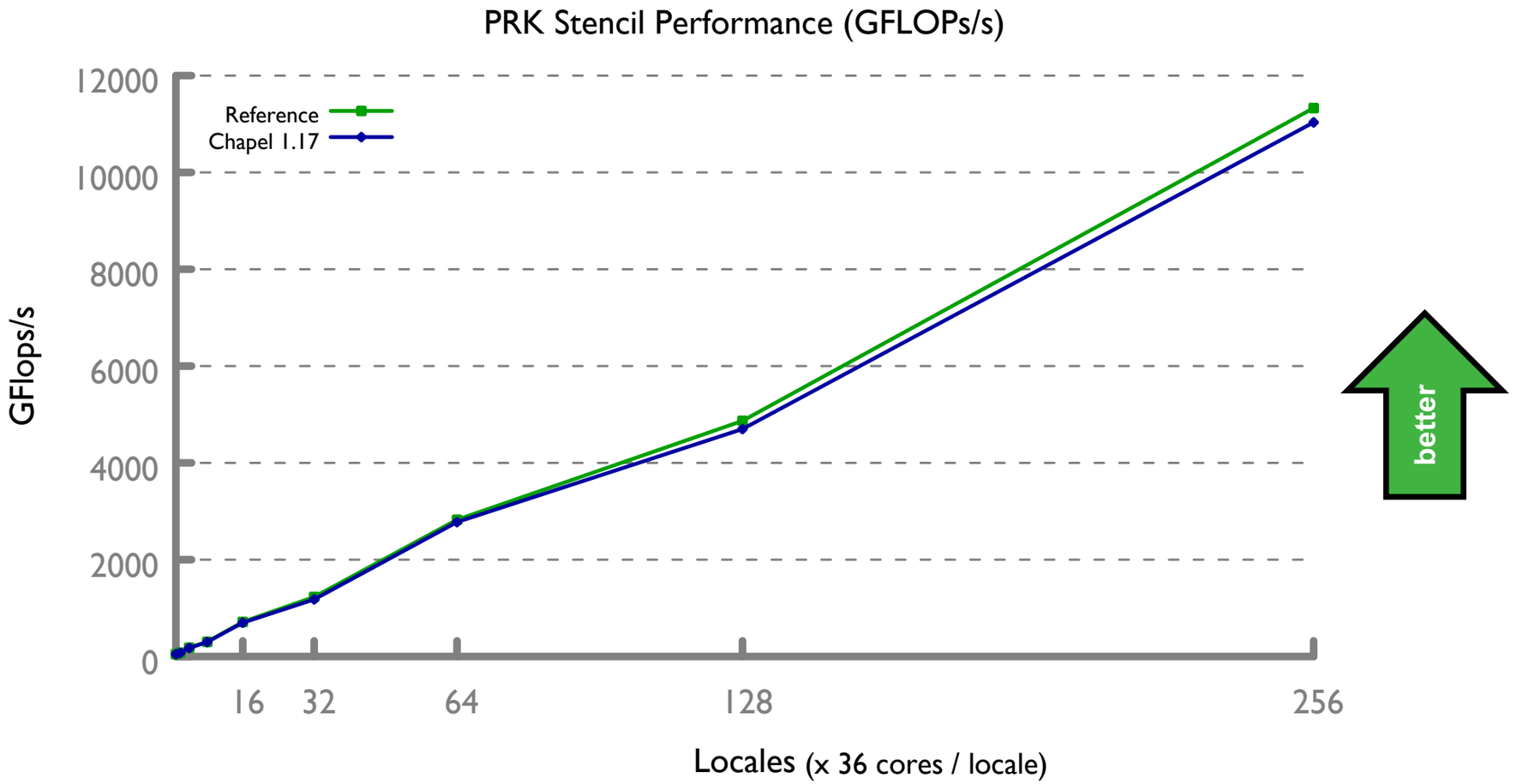
PRK Stencil Performance: Chapel Then



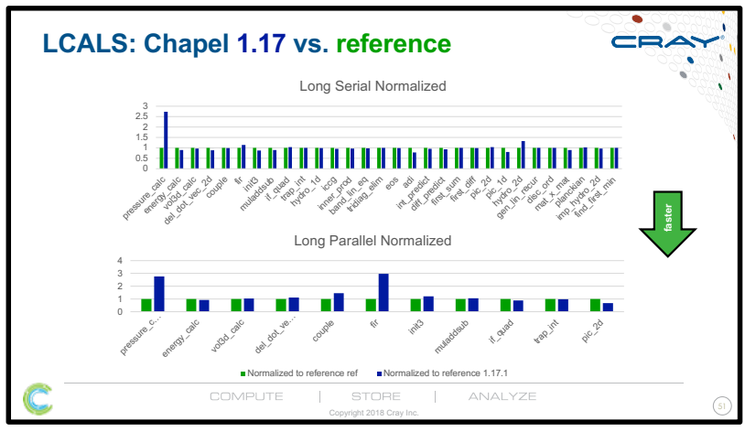
PRK Stencil Performance: Chapel Then vs. Now



PRK Stencil Performance: Chapel Now vs. ref

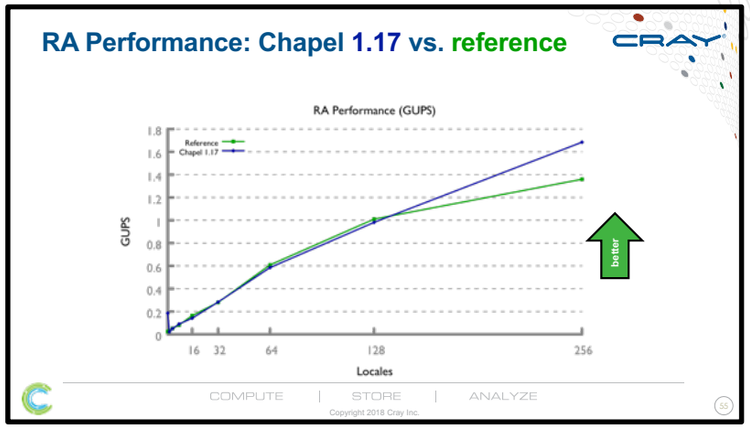


HPC Patterns: Chapel Now vs. reference



LCALS

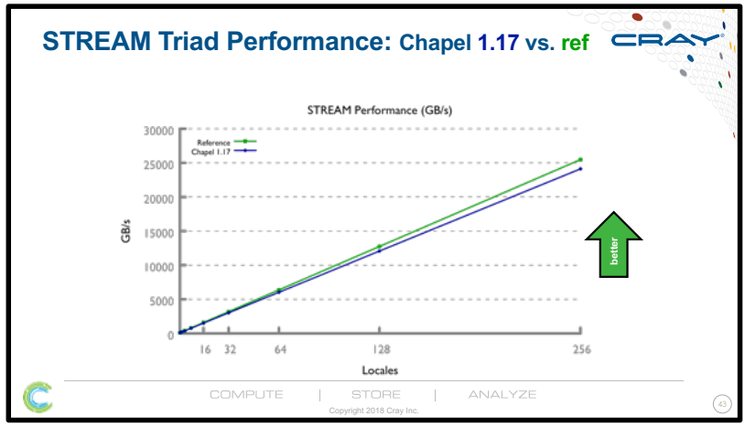
HPCC RA

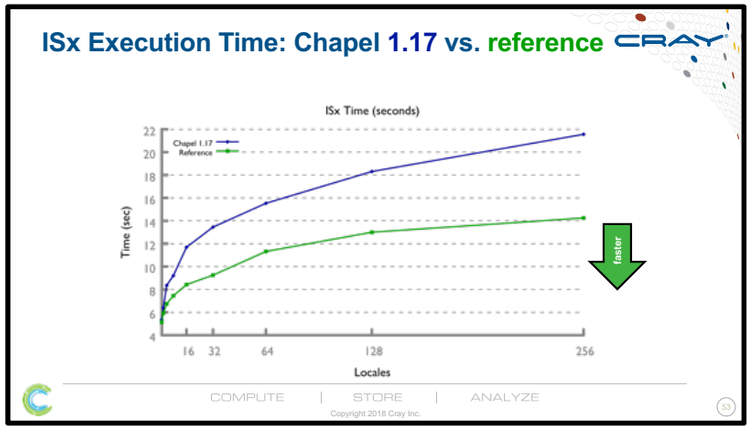


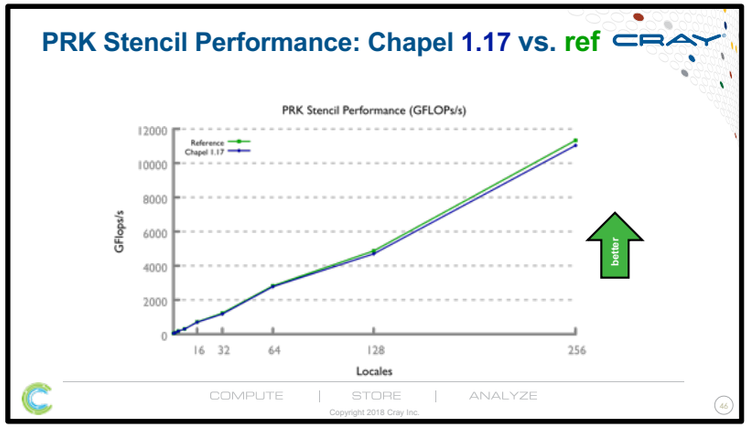
STREAM Triad

ISx

PRK Stencil

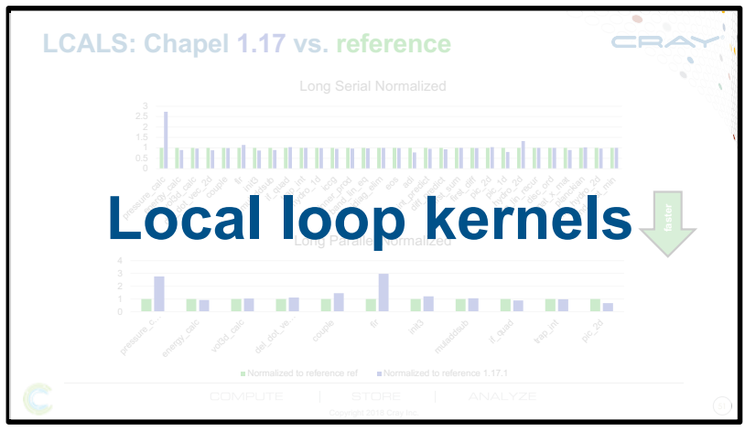






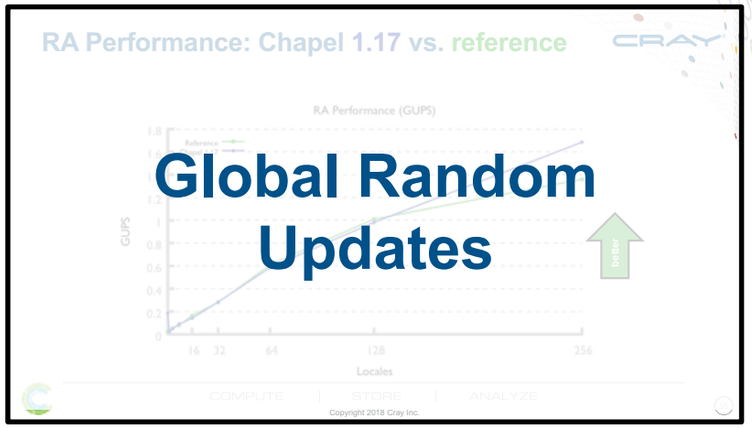
Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

HPC Patterns: Chapel Now vs. reference



LCALS

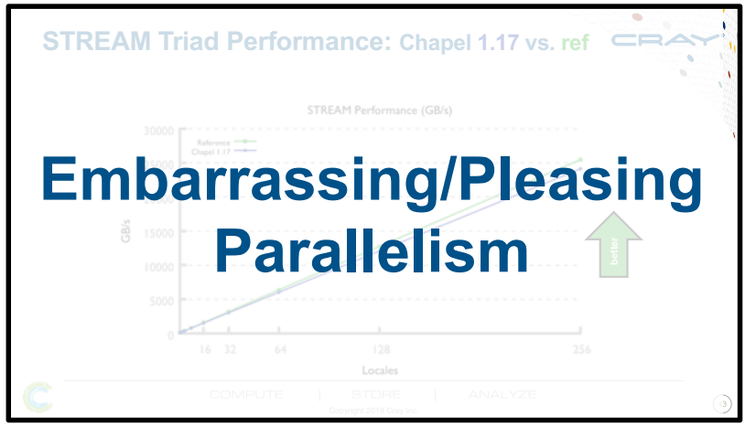
HPCC RA

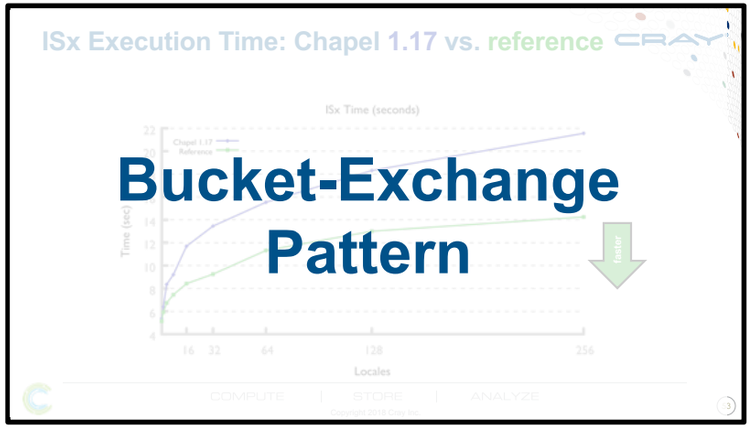


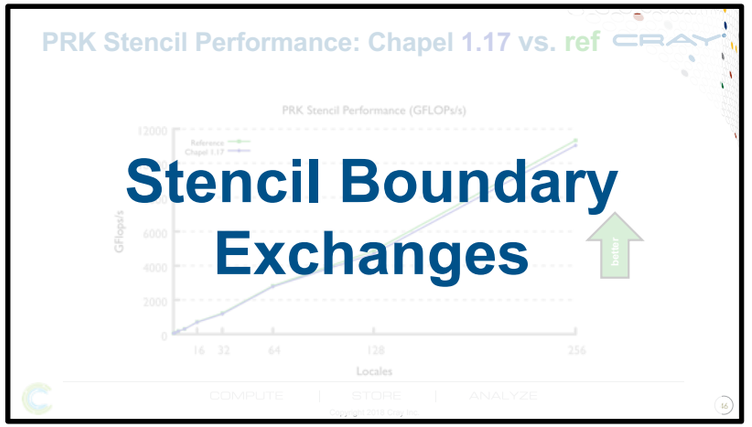
STREAM Triad

ISx

PRK Stencil

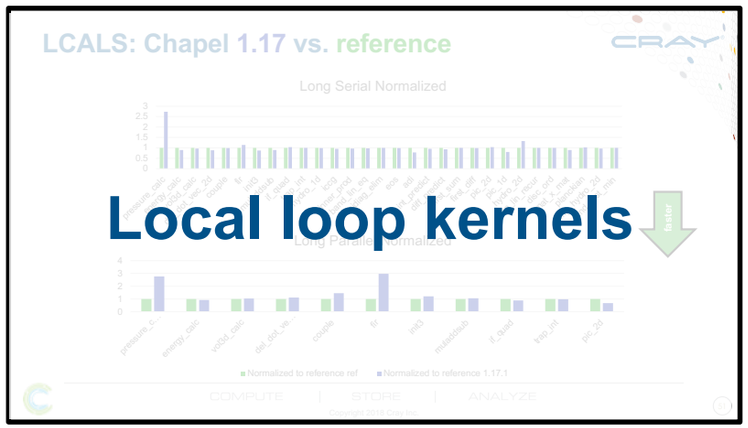






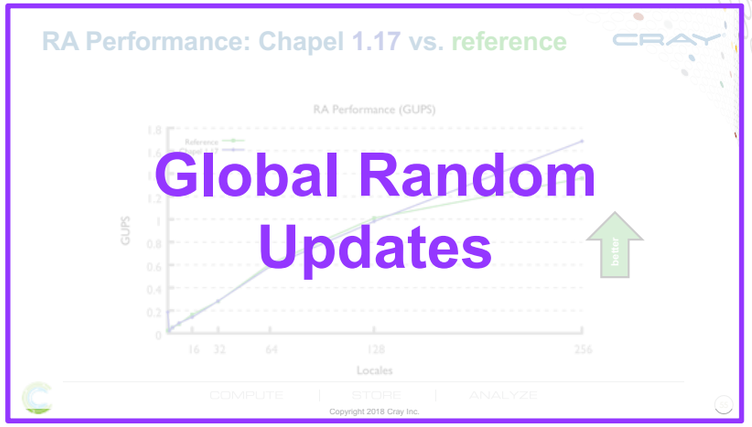
Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

HPC Patterns: Chapel Now vs. reference



LCALS

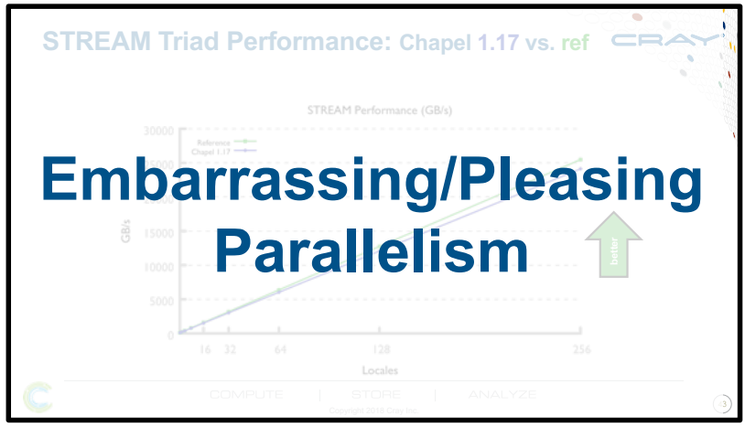
HPCC RA

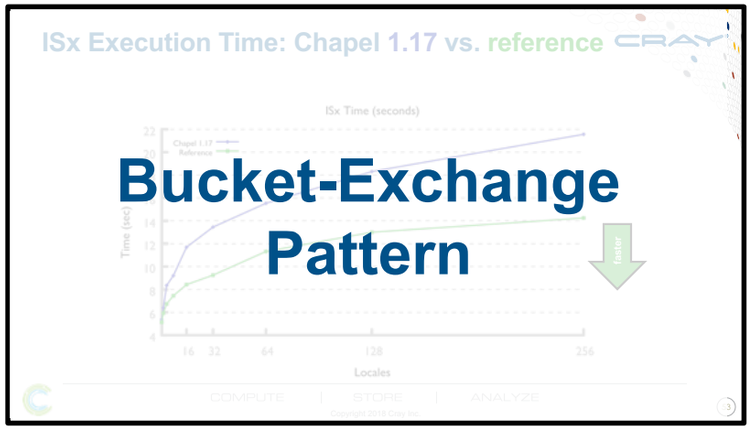


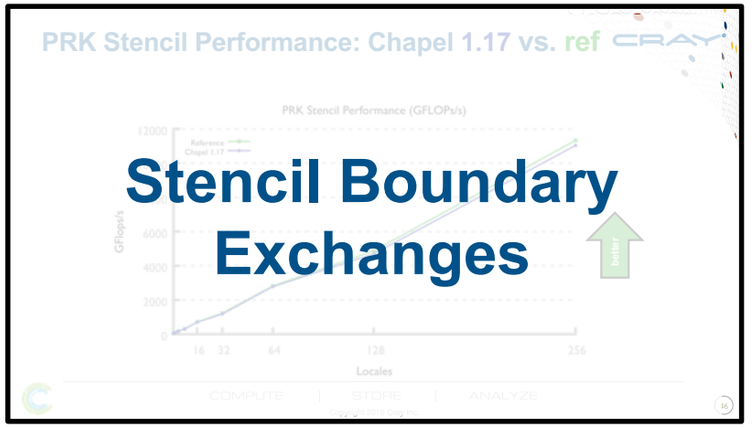
STREAM Triad

ISx

PRK Stencil





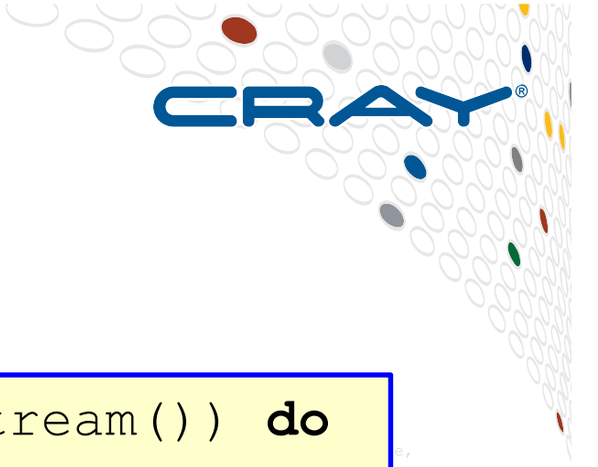


Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

HPCC Random Access 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);
```

HPCC Random Access Kernel: MPI



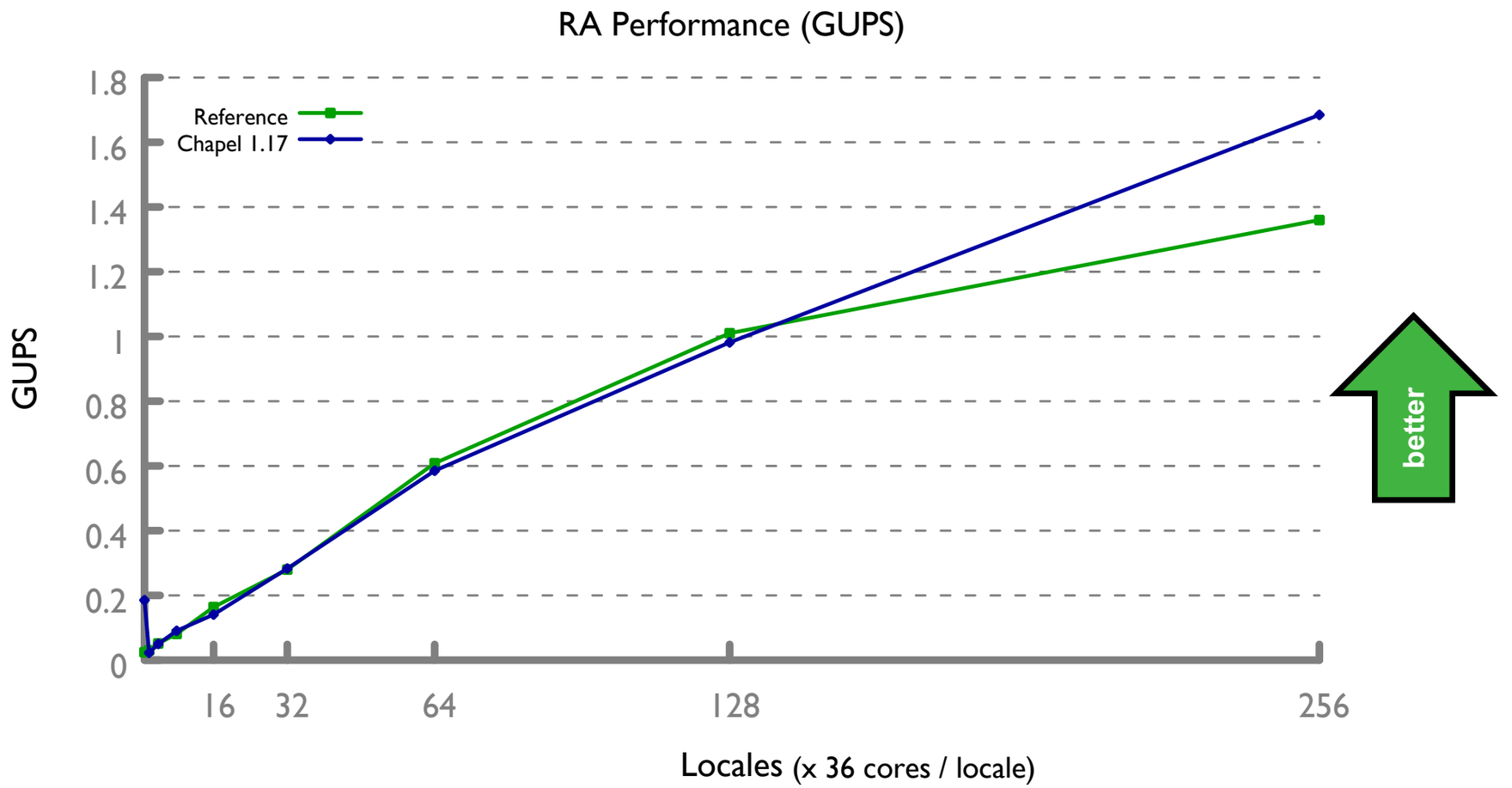
Chapel Kernel

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

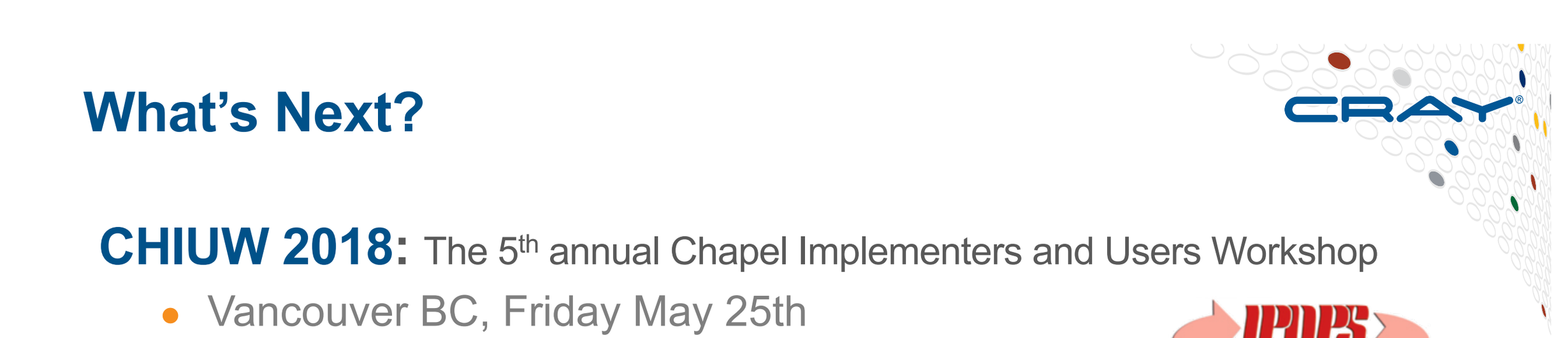
MPI Comment

```
/* 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;
 *     }
 */
```

RA Performance: Chapel Now vs. reference



What's Next?



CHIUW 2018: The 5th annual Chapel Implementers and Users Workshop

- Vancouver BC, Friday May 25th
- Details: https://chapel-lang.org/CHIUW2018.html

Chapel's college years: plans for 2018-2021

- Further Performance and Scalability Improvements
- Libfabric/OFI Support
- GPU Support
- Cloud Support
- Chapel AI

The Chapel Team at Cray (May 2018)

Chapel Community Partners

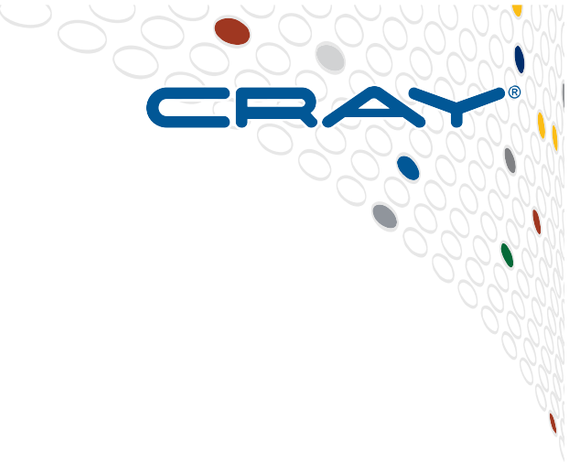
Lawrence Berkeley
National Laboratory

(and several others…)

https://chapel-lang.org/collaborations.html

Summary

- **Chapel's made huge progress over the past five years**
- **Ready for use in production***
- **Open to collaborations**
  - Plenty of research questions remain

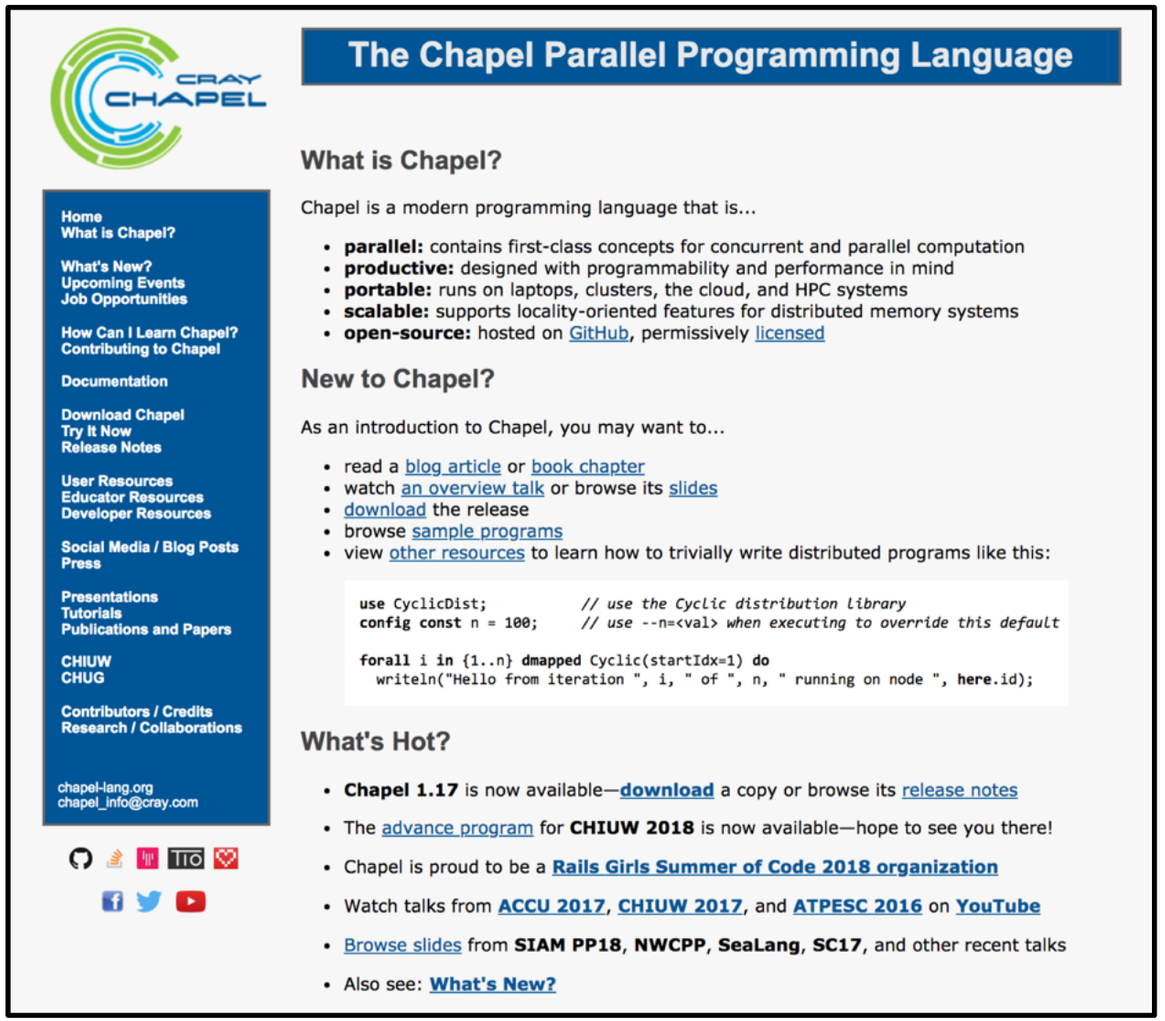


Chapel Resources

Chapel Central

https://chapel-lang.org

- downloads
- documentation
- resources
- presentations
- papers



The Chapel Parallel Programming Language

Home
What is Chapel?
What's New?
Upcoming Events
Job Opportunities
How Can I Learn Chapel?
Contributing to Chapel
Documentation
Download Chapel
Try It Now
Release Notes
User Resources
Educator Resources
Developer Resources
Social Media / Blog Posts
Press
Presentations
Tutorials
Publications and Papers
CHIUW
CHUG
Contributors / Credits
Research / Collaborations
chapel-lang.org
chapel_info@cray.com

What is Chapel?

Chapel is a modern programming language that is...

- **parallel:** contains first-class concepts for concurrent and parallel computation
- **productive:** designed with programmability and performance in mind
- **portable:** runs on laptops, clusters, the cloud, and HPC systems
- **scalable:** supports locality-oriented features for distributed memory systems
- **open-source:** hosted on GitHub, permissively licensed

New to Chapel?

As an introduction to Chapel, you may want to...

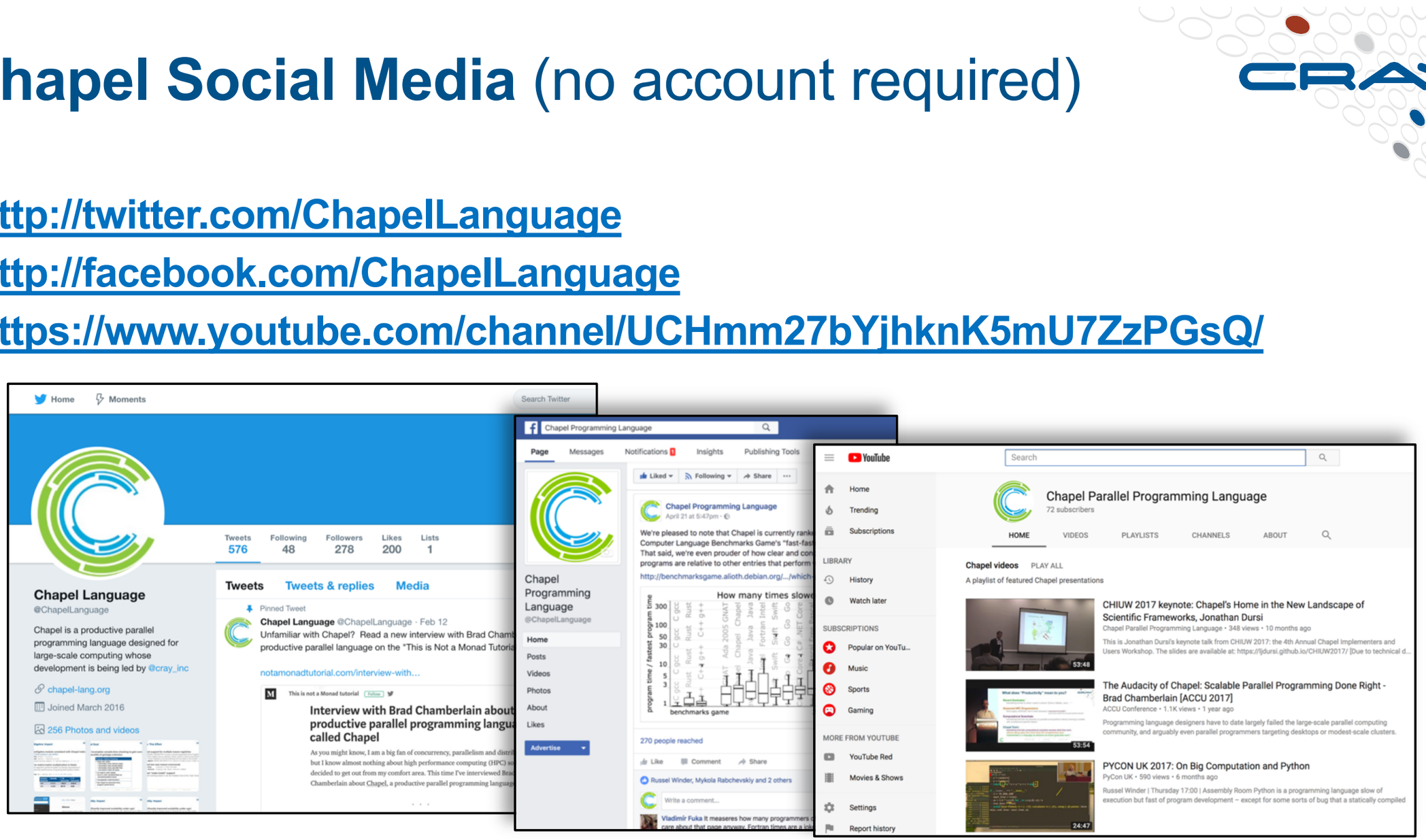
- read a blog article or book chapter
- watch an overview talk or browse its slides
- download the release
- browse sample programs
- view other resources to learn how to trivially write distributed programs like this:

```
use CyclicDist;           // use the Cyclic distribution library
config const n = 100;     // use --n=<val> when executing to override this default

forall i in {1..n} dmapped Cyclic(startIdx=1) do
  writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What's Hot?

- **Chapel 1.17** is now available—**download** a copy or browse its release notes
- The advance program for **CHIUW 2018** is now available—hope to see you there!
- Chapel is proud to be a **Rails Girls Summer of Code 2018 organization**
- Watch talks from **ACCU 2017**, **CHIUW 2017**, and **ATPESC 2016** on **YouTube**
- Browse slides from **SIAM PP18**, **NWCPP**, **SeaLang**, **SC17**, and other recent talks
- Also see: **What's New?**

Chapel Social Media (no account required)



http://twitter.com/ChapelLanguage

http://facebook.com/ChapelLanguage

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

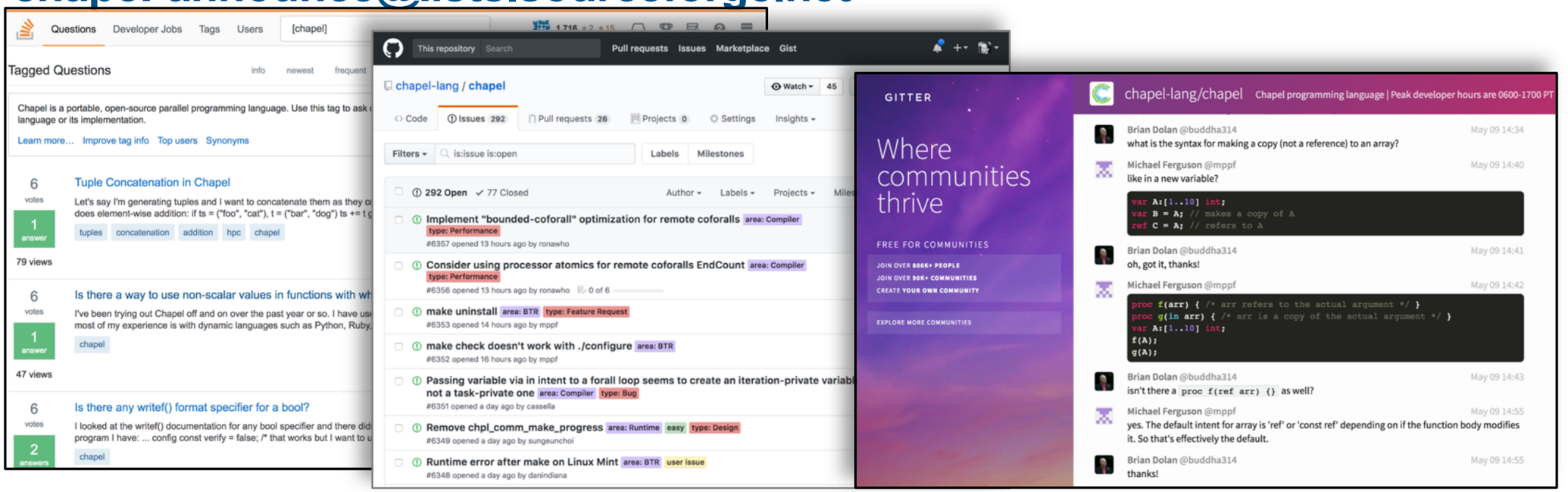
Chapel Community

https://stackoverflow.com/questions/tagged/chapel

https://github.com/chapel-lang/chapel/issues

https://gitter.im/chapel-lang/chapel

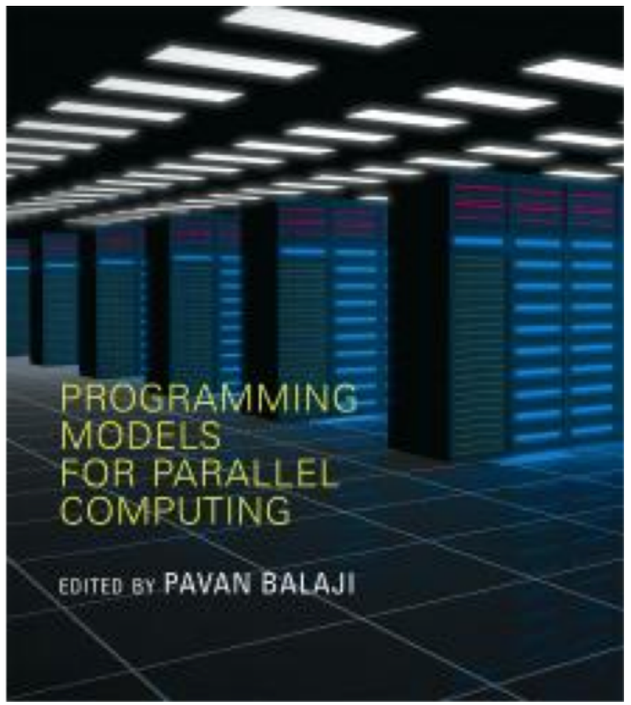
chapel-announce@lists.sourceforge.net



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 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 (as of 2017)*

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*

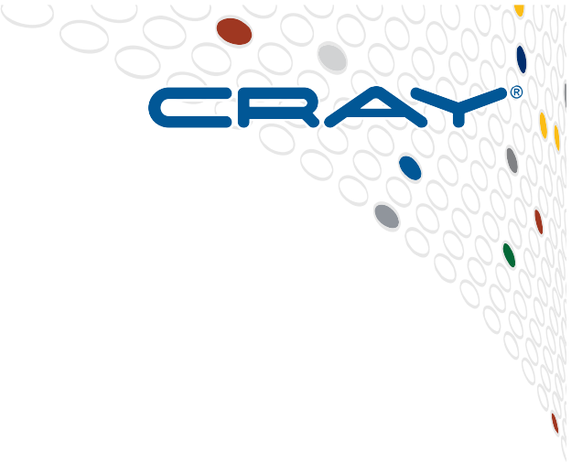
Why Chapel? **(parts 1, 2, 3)**, Cray Blog, Jun-Oct 2014.

- *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*

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
- Gitter (chapel-lang/chapel): community chat with archives
- chapel-users@lists.sourceforge.net: user discussions

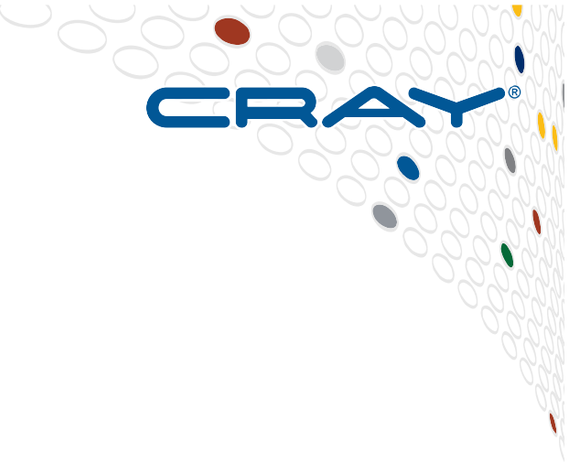
Discuss Chapel development

- chapel-developers@lists.sourceforge.net: developer discussions
- GitHub issues for chapel-lang/chapel: for feature requests, design discussions

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.

CRAY

THE SUPERCOMPUTER COMPANY