Chapel's New Adventures in Data Locality

Brad Chamberlain
Chapel Team, Cray Inc.
August 2, 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.

What is Chapel?

Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort

Goals:

- Support general parallel programming at scale
- Make parallel programming 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]

The Chapel Team at Cray (May 2017)

The Broader Chapel Community (a subset)

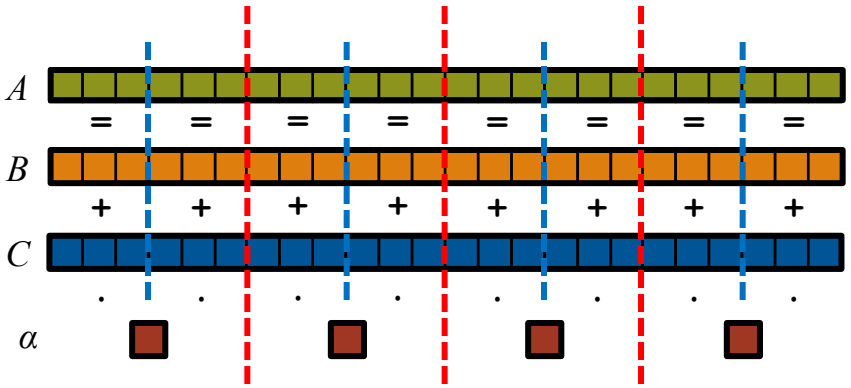
Lawrence Berkeley
National Laboratory

http://chapel.cray.com/collaborations.html

Scalable Parallel Programming Concerns

Typical Chapel programmers should focus on:

- *Parallelism:* What should execute simultaneously?
- *Locality:* Where should those tasks execute? their data reside?



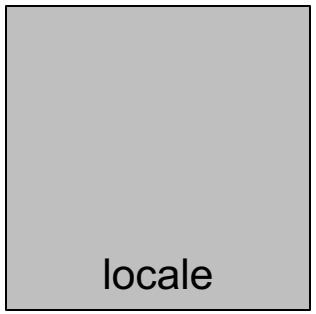
Outline

- ✓ What's Chapel?
- ➢ **Classic Chapel Concepts for Locality ('CCC's)**
- Three Recent Locality Endeavors ("New Adventures")
- Wrap-up

CCC #1: Locales

locale: **Chapel type/values representing architectural locality**

- (think “compute node”)



CCC #1: Locales

***locale:* Chapel type/values representing architectural locality**

- (think “compute node”)
- Chapel automatically provides a 1D array of locales:

```
const Locales: [0..#numLocales] locale;
```

CCC #2: on-clauses

on-clause: **Moves the current task to the specified locale**

CCC #2: on-clauses

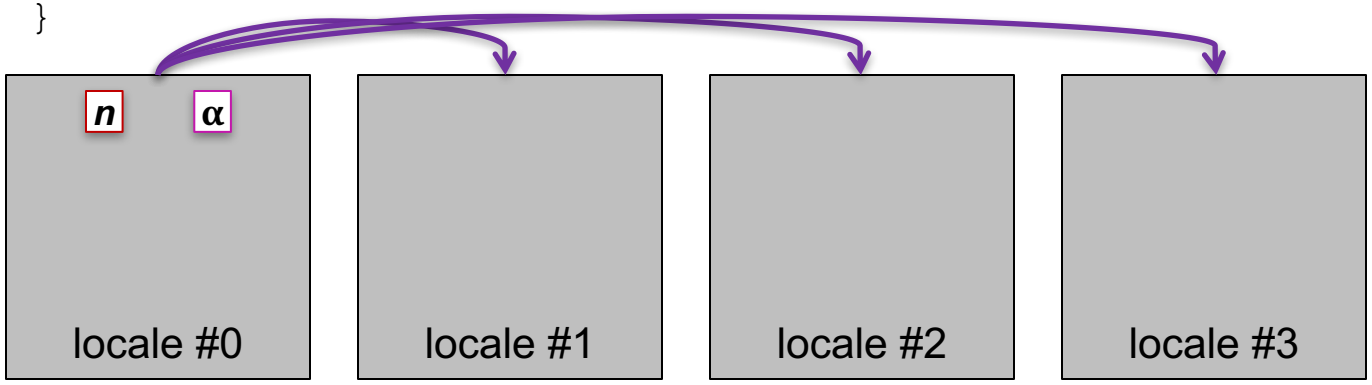
on-clause: **Moves the current task to the specified locale**

```
// programs begin execution as a single task on locale #0
config const n = computeLocalProblemSize(),
             alpha = 0.5;
```

CCC #2: on-clauses

on-clause: Moves the current task to the specified locale

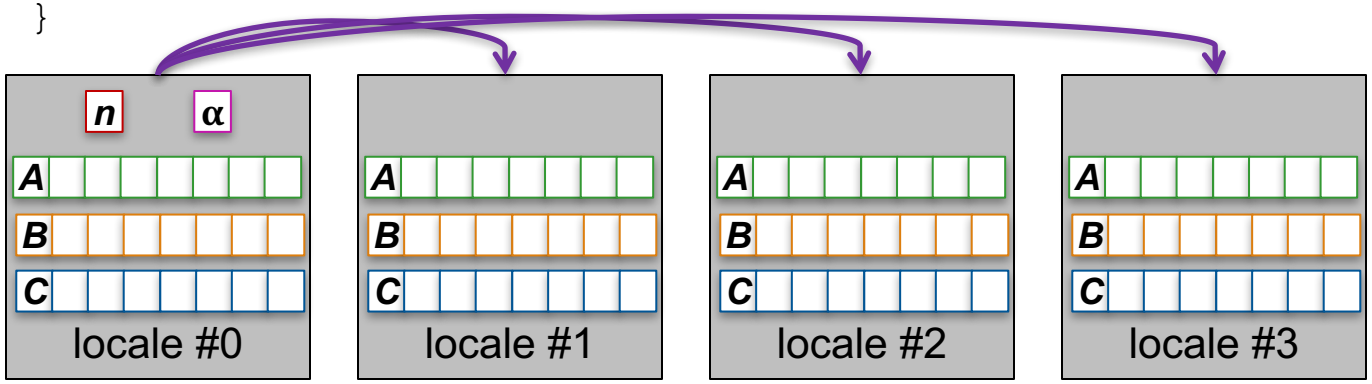
```
// programs begin execution as a single task on locale #0
config const n = computeLocalProblemSize(),
             alpha = 0.5;
coforall loc in Locales do   // creates a task per locale
  on loc {                   // moves the task to its locale


}
```



CCC #2: on-clauses

on-clause: Moves the current task to the specified locale

```
// programs begin execution as a single task on locale #0
config const n = computeLocalProblemSize(),
             alpha = 0.5;
coforall loc in Locales do   // creates a task per locale
  on loc {                   // moves the task to its locale
    var A, B, C: [1..n] real;
    A = B + alpha * C;
  }
```

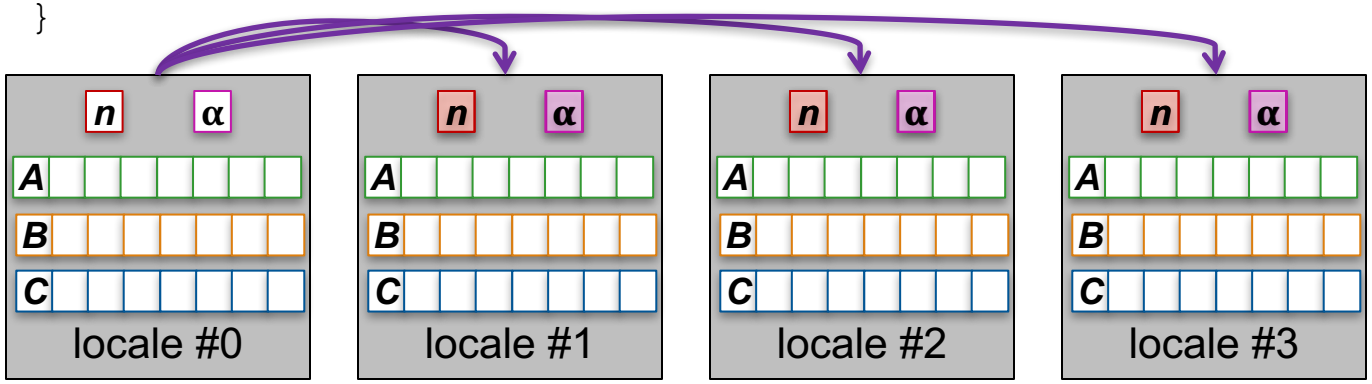


(conceptual view)

CCC #2: on-clauses

on-clause: Moves the current task to the specified locale

```
// programs begin execution as a single task on locale #0
config const n = computeLocalProblemSize(),
             alpha = 0.5;
coforall loc in Locales do   // creates a task per locale
  on loc {                   // moves the task to its locale
    var A, B, C: [1..n] real;
    A = B + alpha * C;
  }
```



(optimized view)

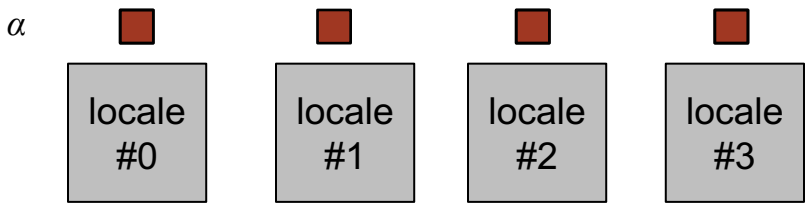
CCC #3: Distributions / Domain Maps

distribution: **Maps *domains* ("index sets") to locales**

CCC #3: Distributions / Domain Maps

distribution: **Maps *domains* ("index sets") to locales**

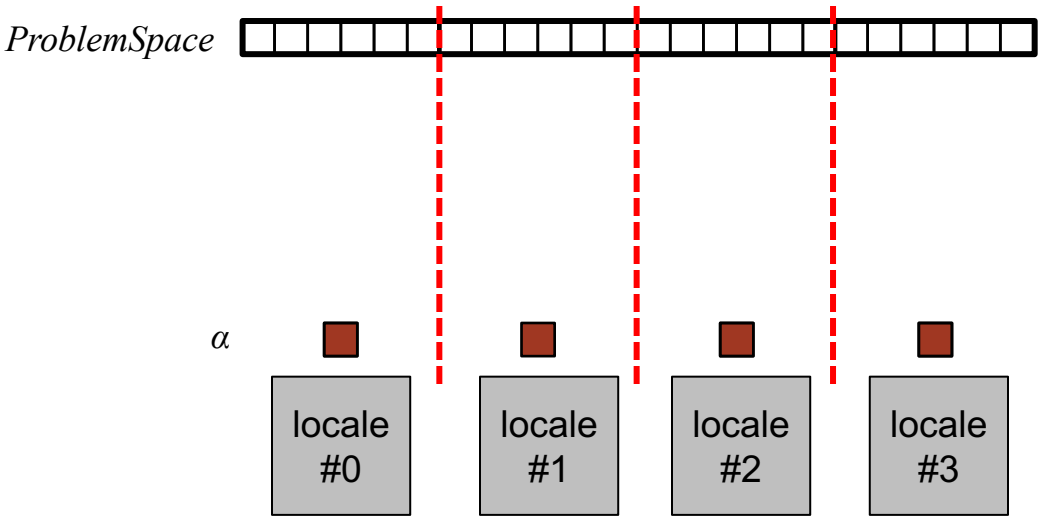
```
config const n = computeGlobalProblemSize(),
             alpha = 0.5;
```



CCC #3: Distributions / Domain Maps

distribution: Maps *domains* ("index sets") to locales

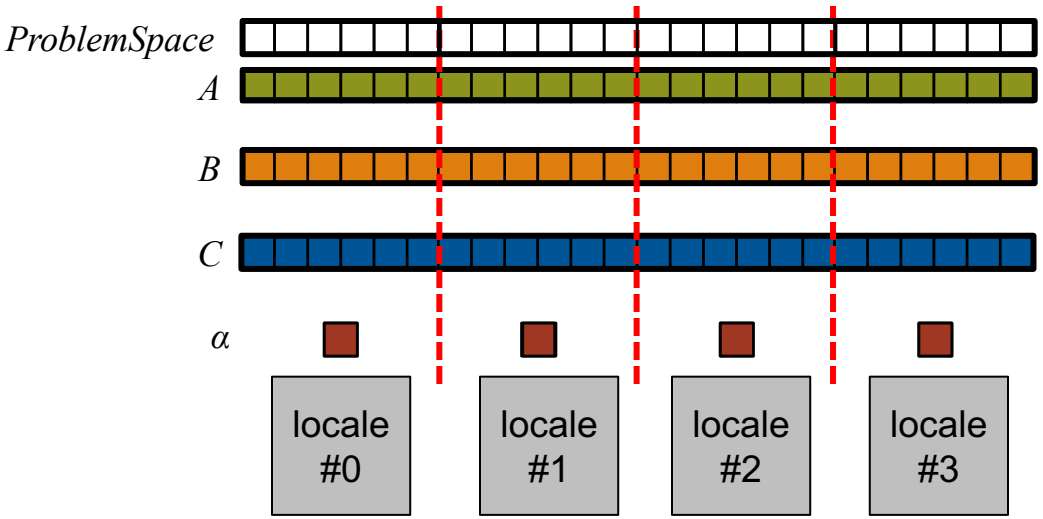
```
config const n = computeGlobalProblemSize(),
             alpha = 0.5;
use BlockDist;
const ProblemSpace = {1..n} dmapped Block(…);
```



CCC #3: Distributions / Domain Maps

distribution: **Maps *domains* ("index sets") to locales**

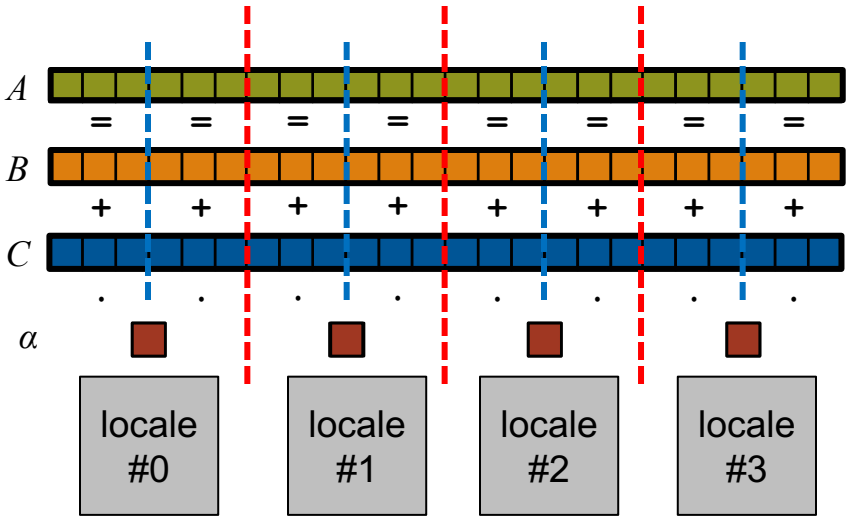
```
config const n = computeGlobalProblemSize(),
             alpha = 0.5;
use BlockDist;
const ProblemSpace = {1..n} dmapped Block(…);
var A, B, C: [ProblemSpace] real;
```



CCC #3: Distributions / Domain Maps

distribution: Maps *domains* ("index sets") to locales

```
config const n = computeGlobalProblemSize(),
             alpha = 0.5;
use BlockDist;
const ProblemSpace = {1..n} dmapped Block(…);
var A, B, C: [ProblemSpace] real;
A = B + alpha * C;
```



CCC #4: User Control over Locality Policies

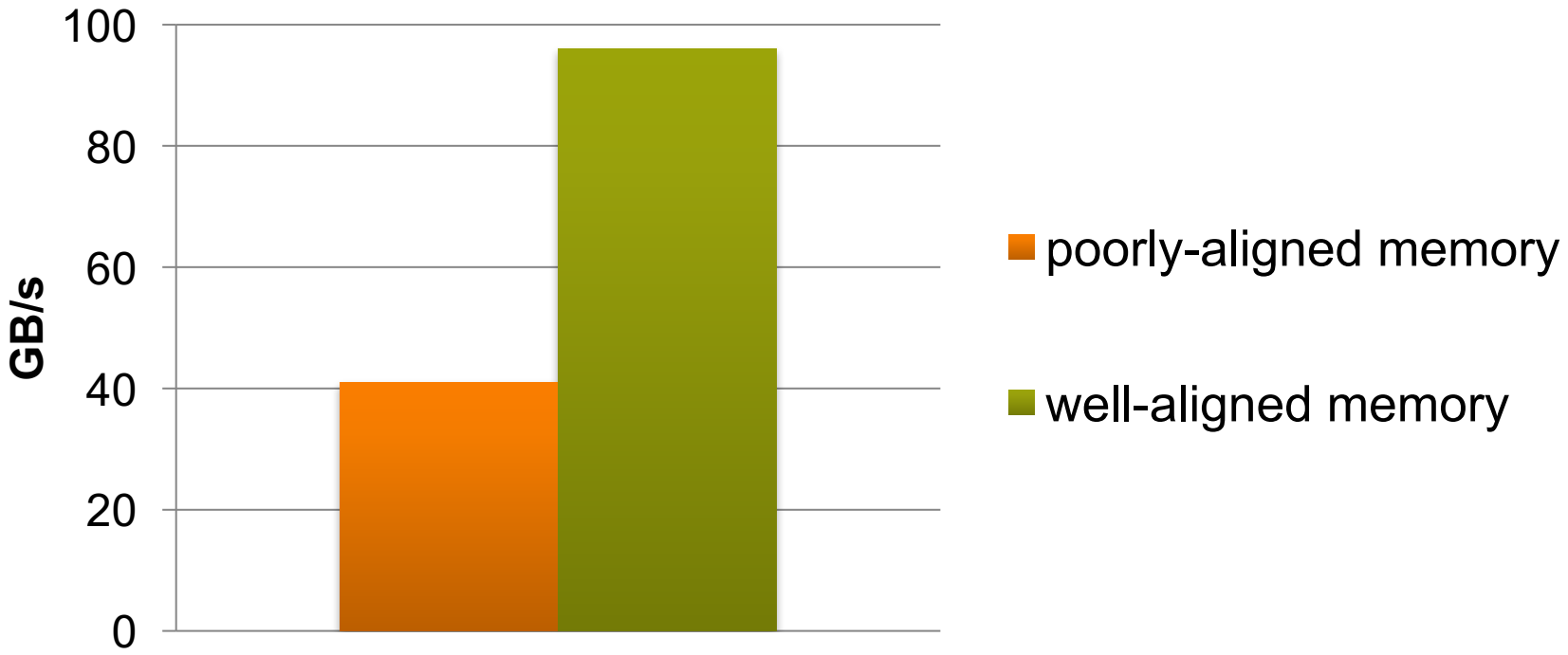
- **In Chapel, users can…**
  - …write their own distributions
    - "How should domains & arrays be mapped to locales and their memories?"
  - …write their own parallel iterators
    - "How should forall-loops be implemented? How many tasks, running where?"
  - …write their own locale models
    - "How should tasks, memory, communication be mapped to the system?"

- **This gives users full control over key locality policies**
  - Moreover, all "built-in" Chapel features are written in this framework

Locality Adventure #1: **NUMA Locale Model**

The Perils of NUMA Obliviousness

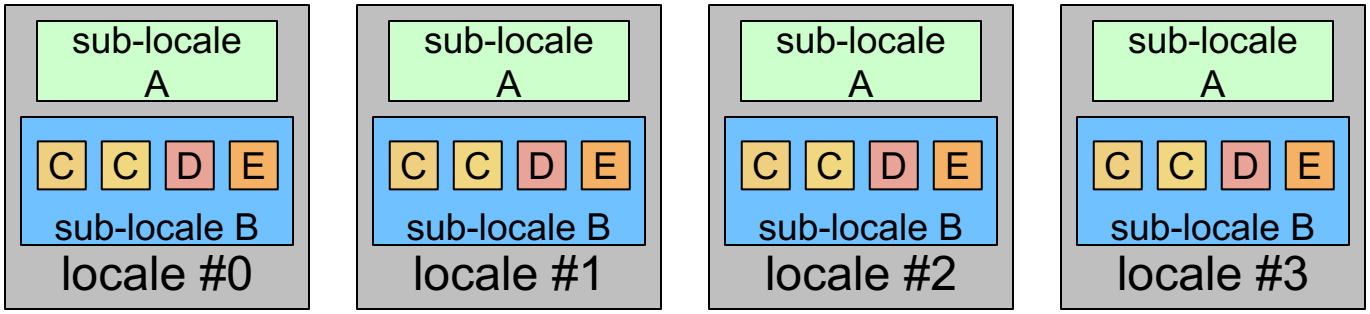
- **Accessing non-NUMA-local memory ⇒ performance hit**
  - e.g., Stream EP on Cray XC w/ 2 NUMA domains per node:



Flat vs. Hierarchical Locales

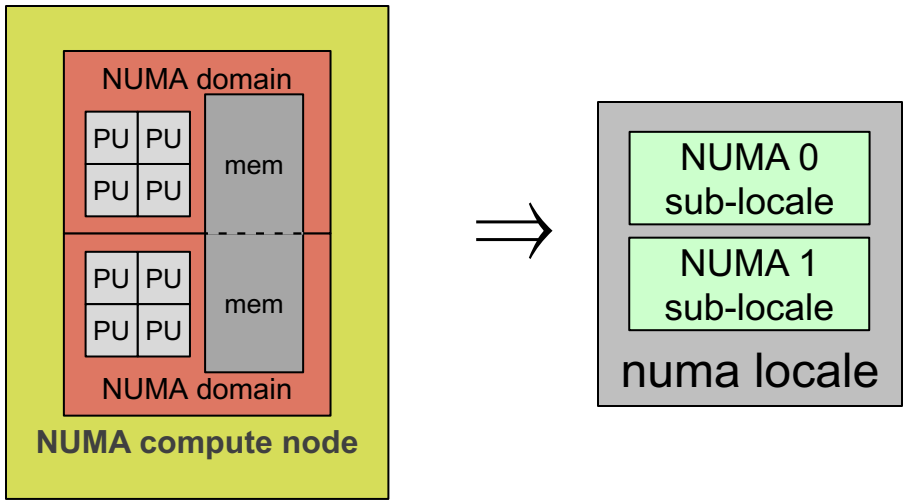
- **Traditionally, Chapel has supported a "flat" locale model**
  - intra-locale decisions are managed on the user's behalf

- **But, users can also write hierarchical locale models**



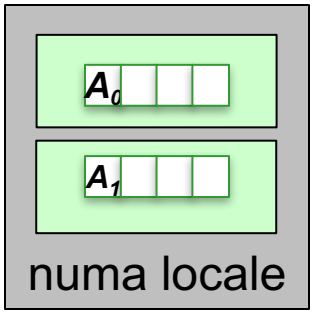
Adventure #1: NUMA locale model

- Created ‘numa’ locale model to describe NUMA nodes

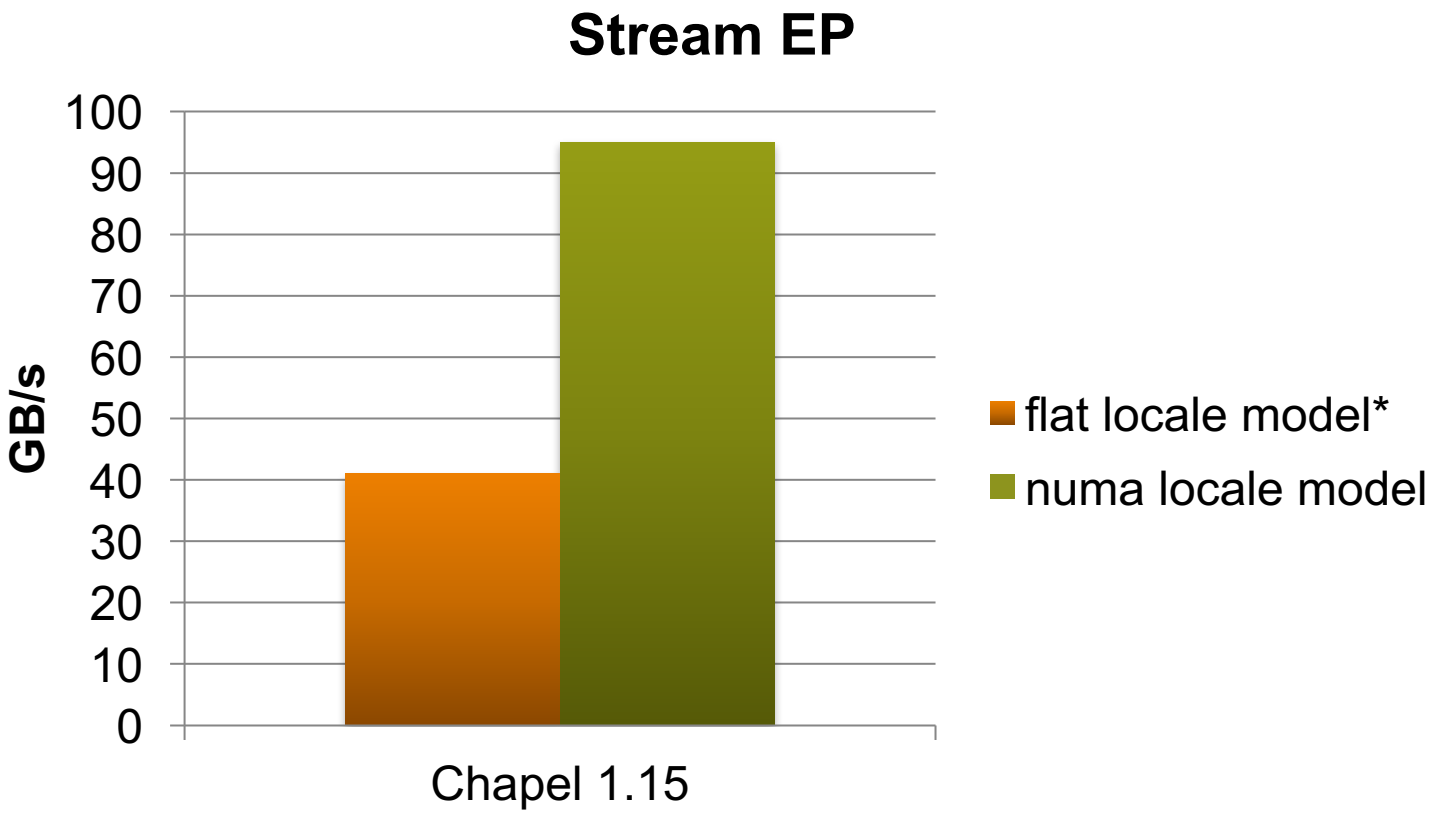


- Also made the default domain map NUMA-aware
  - allocates local arrays using a chunk per sublocale

```
var A: [1..n] real;
```

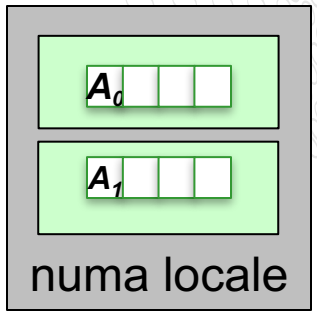


Adventure #1: Positive Impact

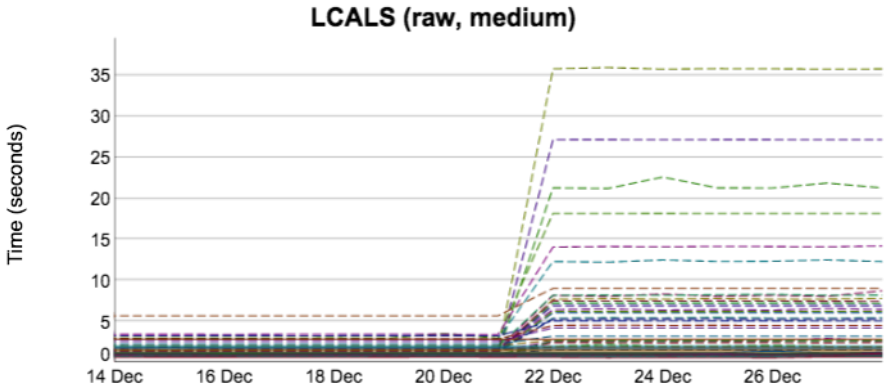


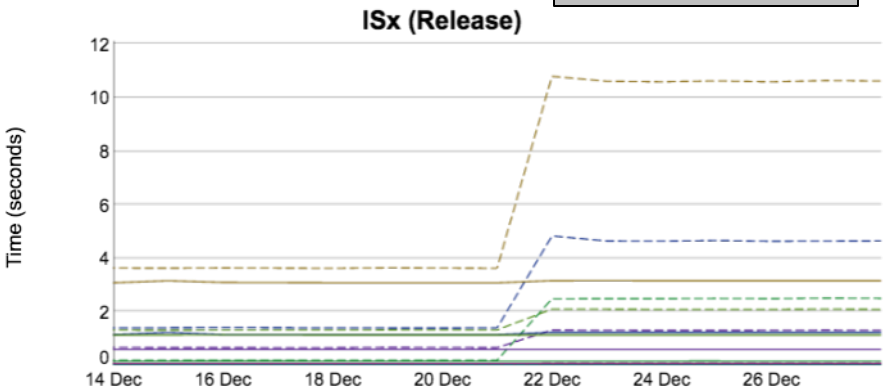
* = ostensibly… we'll come back to this in a few slides

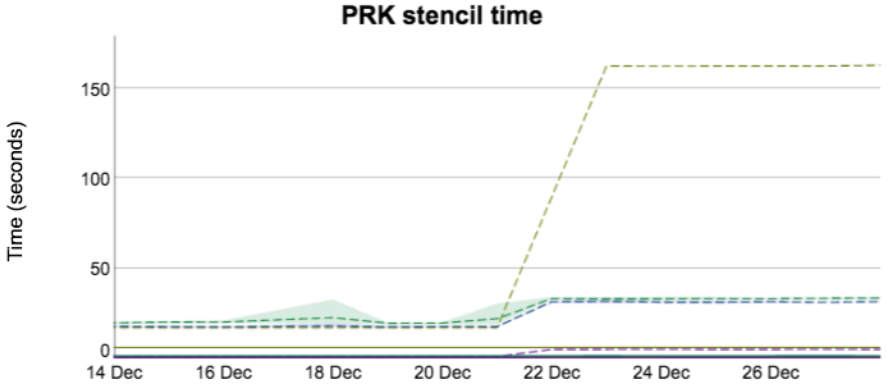
Adventure #1: Negative Impact

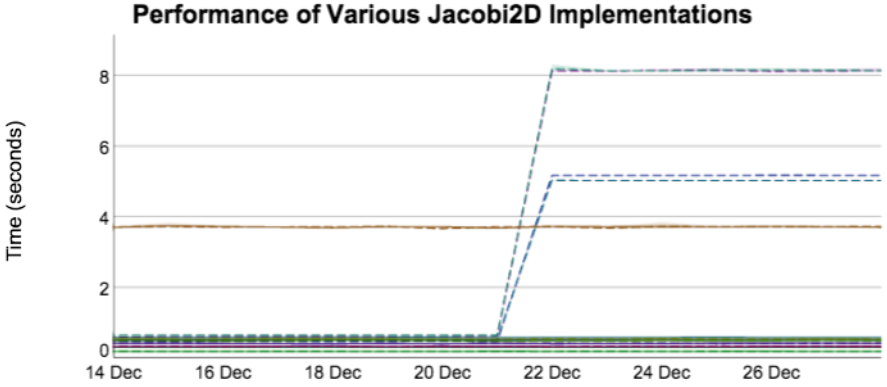


- Array accesses like `A[i]` now require a divide









Adventure #1: Summary

- **The increased array access cost is problematic**
  - We'd like these idioms to all perform equivalently in Chapel:
    - whole-array operations:

```
A = B + alpha * C;
```

    - zippered iteration:

```
forall (a, b, c) in zip(A, B, C) do
  a = b + alpha * c;
```

    - random access:

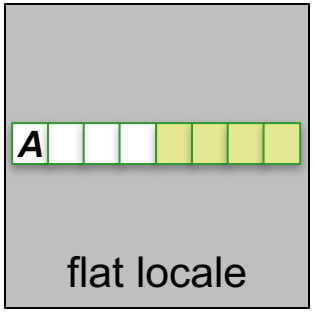
```
forall i in ProblemSpace do
  A[i] = B[i] + alpha * C[i];
```

  - While there are ways to mitigate the overheads, they aren't ideal
    - still not overhead-free (in some approaches)
    - too expensive to implement (in others)
  - So, let's try something else…

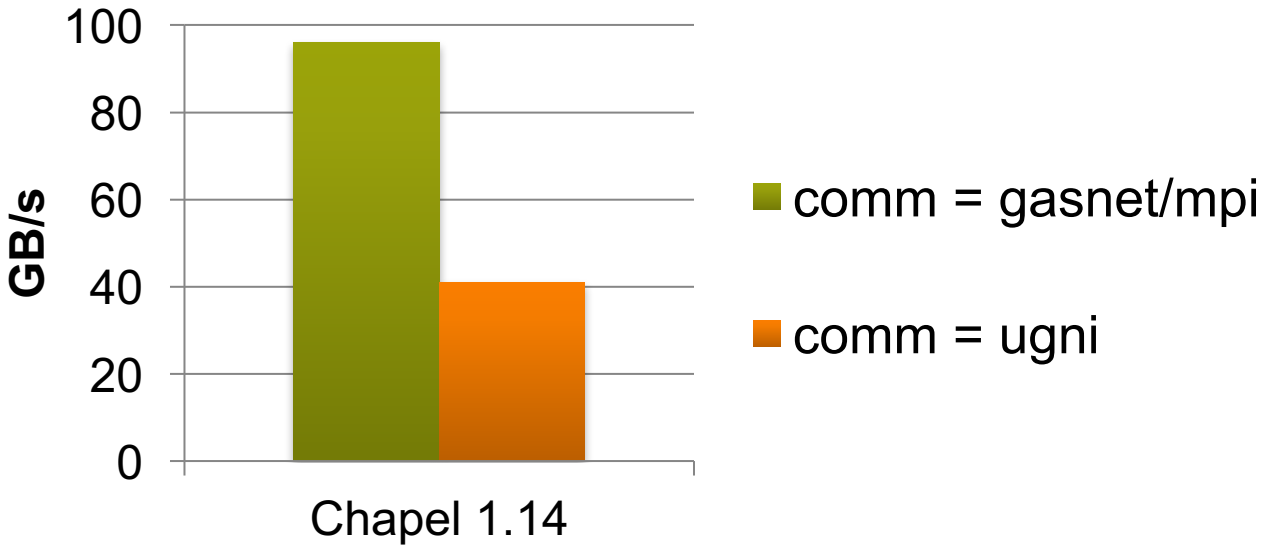
Locality Adventure #2: **PGAS, Networks, & Locality**

Flat Locale Model: Correcting a White Lie

- **I suggested that the flat locale model is NUMA-oblivious**
  - It is, but the default domain map actually is not
    - it distributes array indices using first-touch, heuristically



  - Sometimes results in good performance, but not always:

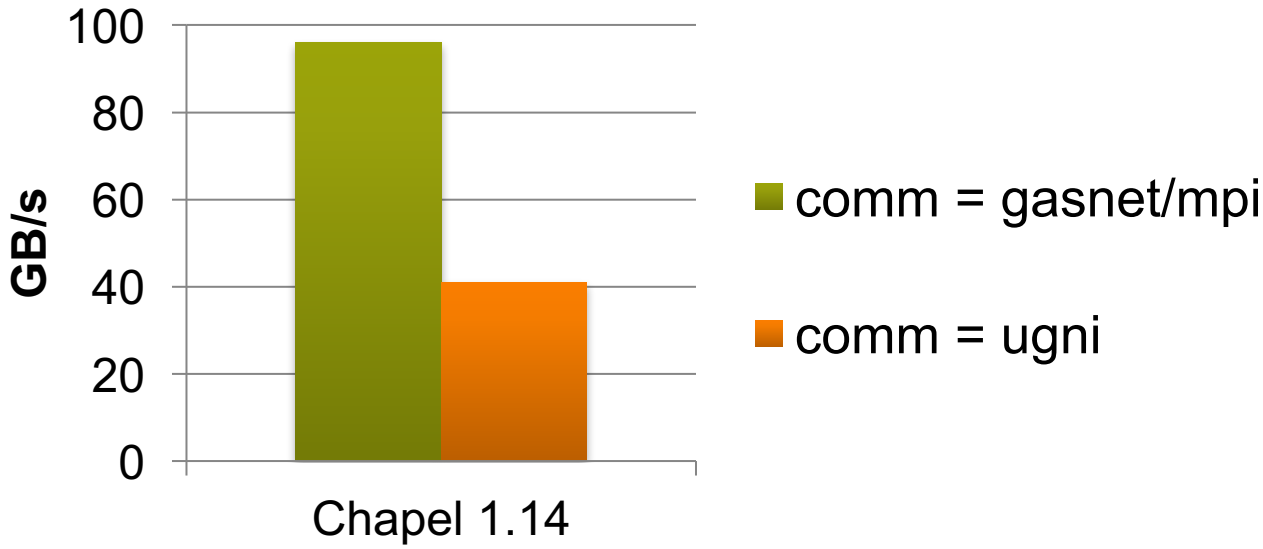


Flat Locale Model: Why does GASNet/MPI win?

- **Chapel usually performs best when using ugni**
  - leverages Cray network capabilities
  - matches Chapel's PGAS features well

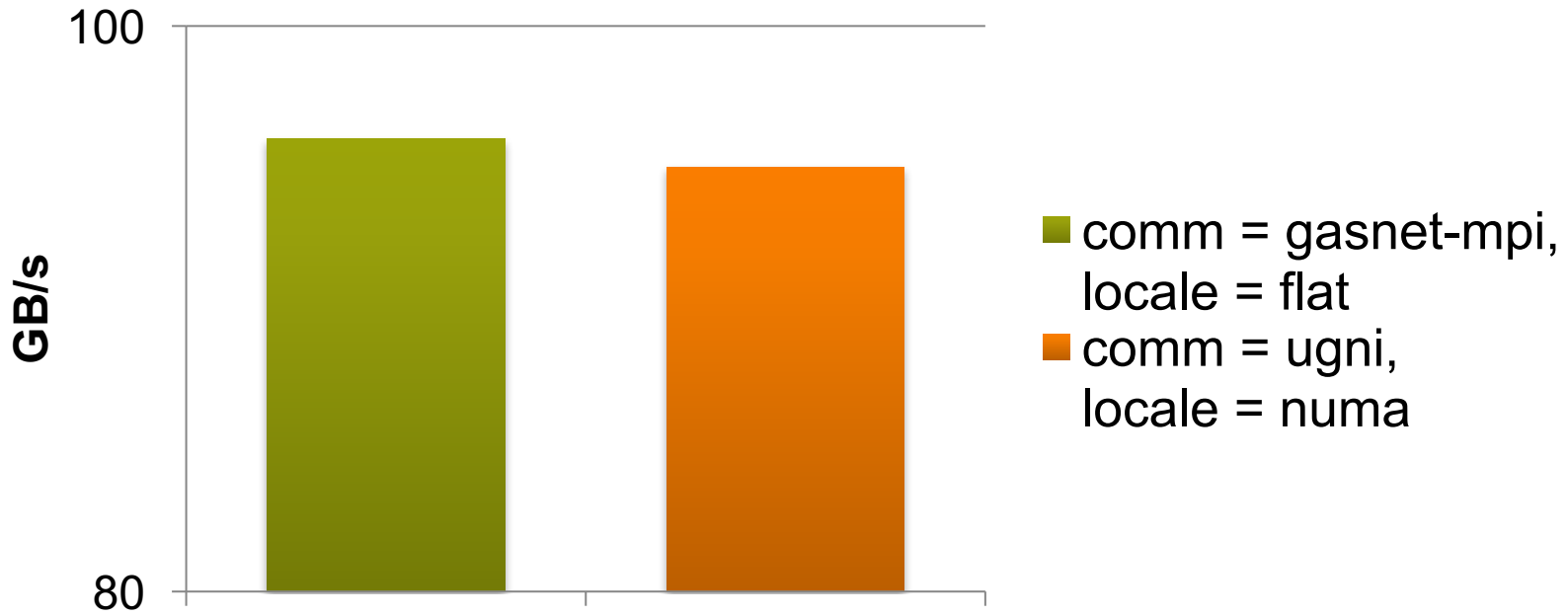
  Q: why not here, when no communication is used?

  A: PGAS-based network registration of heap at program startup
    - serves as first-touch, pinning all memory to NUMA domain 0
    - lack of communication magnifies memory-oriented bottlenecks



NUMA locale model and network registration

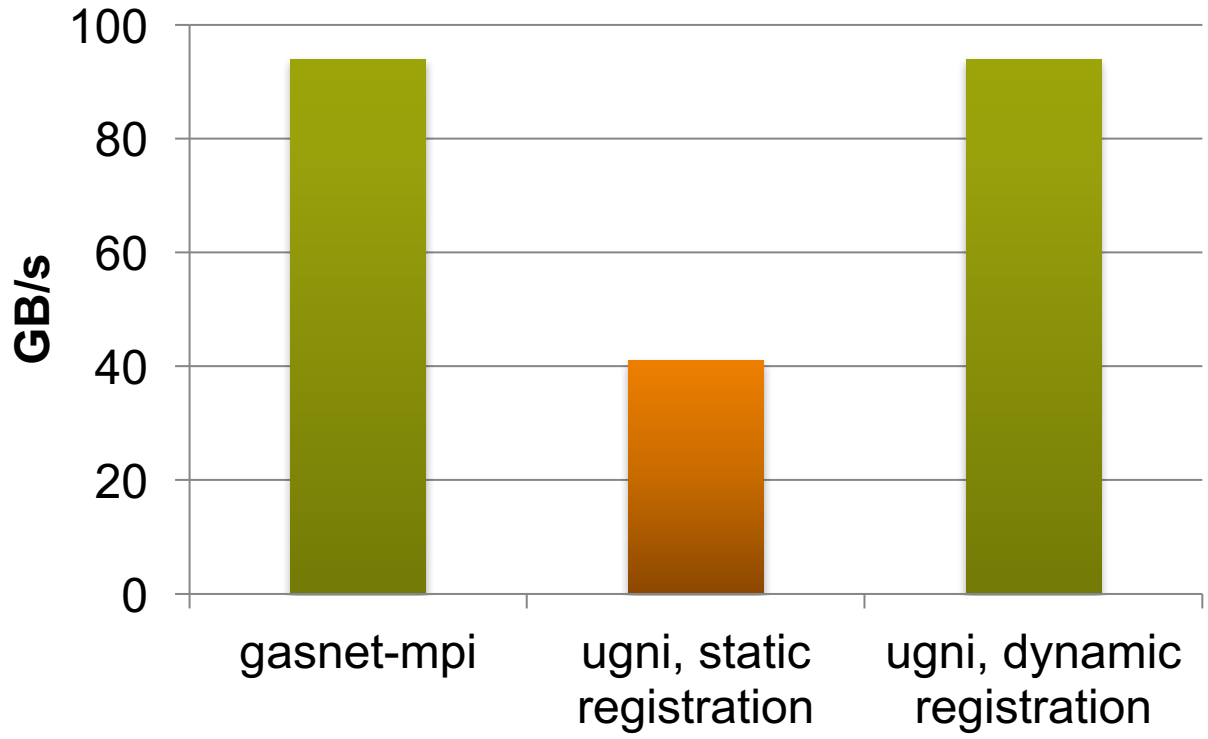
- **For 'numa', each sublocale registers its own local heap**
  - thus, this is one approach to addressing the problem

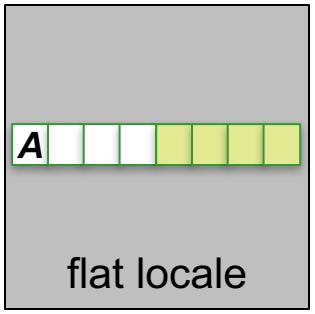


  - but, it introduces the aforementioned overheads for random access

Adventure #2: Dynamic Memory Registration

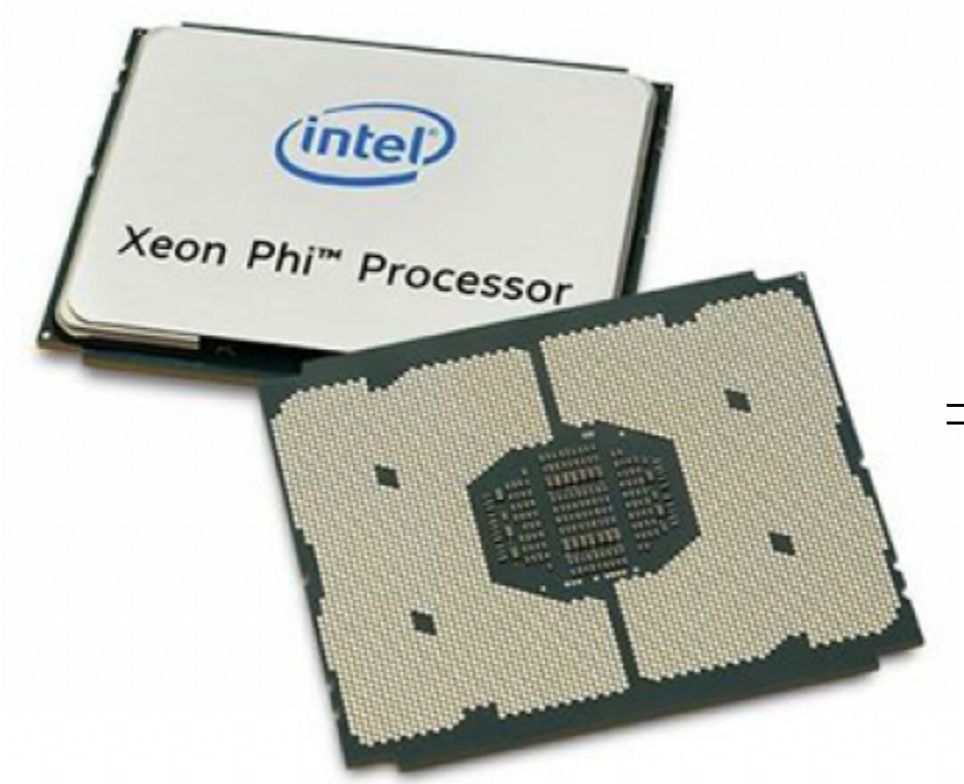
- **Register array memory with network at allocation time**
  - heuristically, divide array into approximately equal # of pages
- **Impact: Restores performance for ugni:**





Locality Adventure #3: **Intel Xeon Phi ("KNL") HBM**

Adventure #3: KNL Locale Model



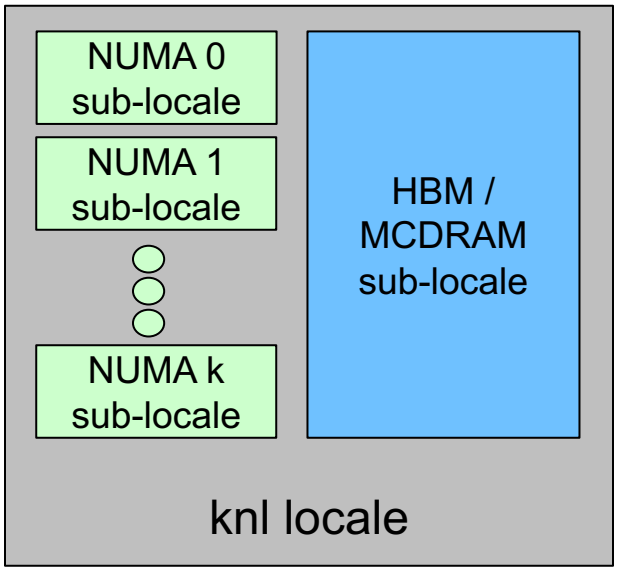


Image Source: https://newsroom.intel.com/press-kits/intel-xeon-phi-processor-family/

KNL Locale Model: Usage and Status

- **Chapel can target KNL's MCDRAM via normal on-clauses**
  - accessor methods expose memory-based sub-locales
  - methods implemented across all standard locale models for portability

```
on here.highBandwidthMemory() {
  x = new myClass();       // placed in MCDRAM
  ...
  on here.defaultMemory() {
    y = new myClass();     // placed in DDR
    ...
  }
}
```

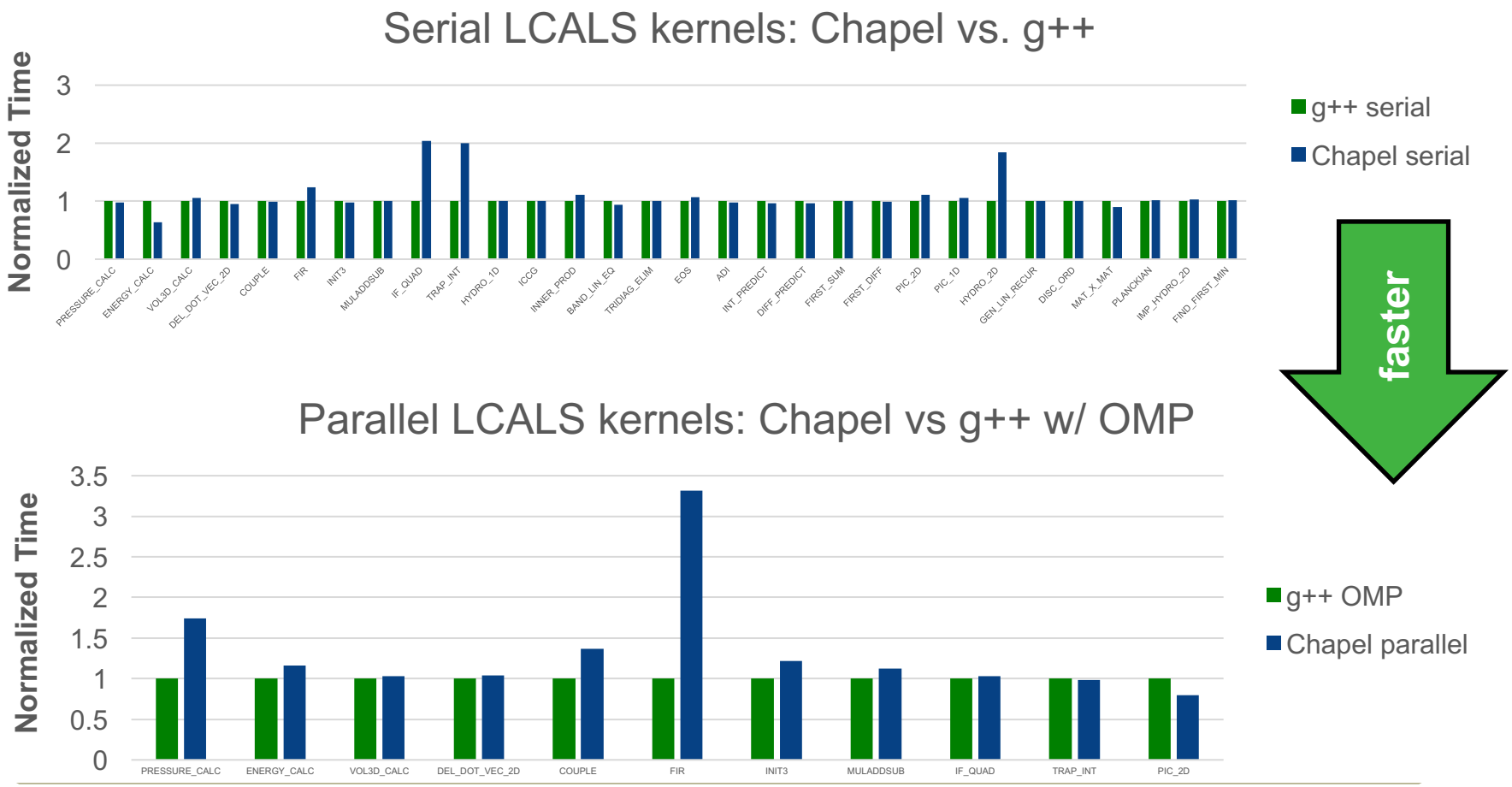
Status: Supported as of Chapel 1.15

- no performance results to report at this time
- next step: improve support for memory introspection ("if I have…")

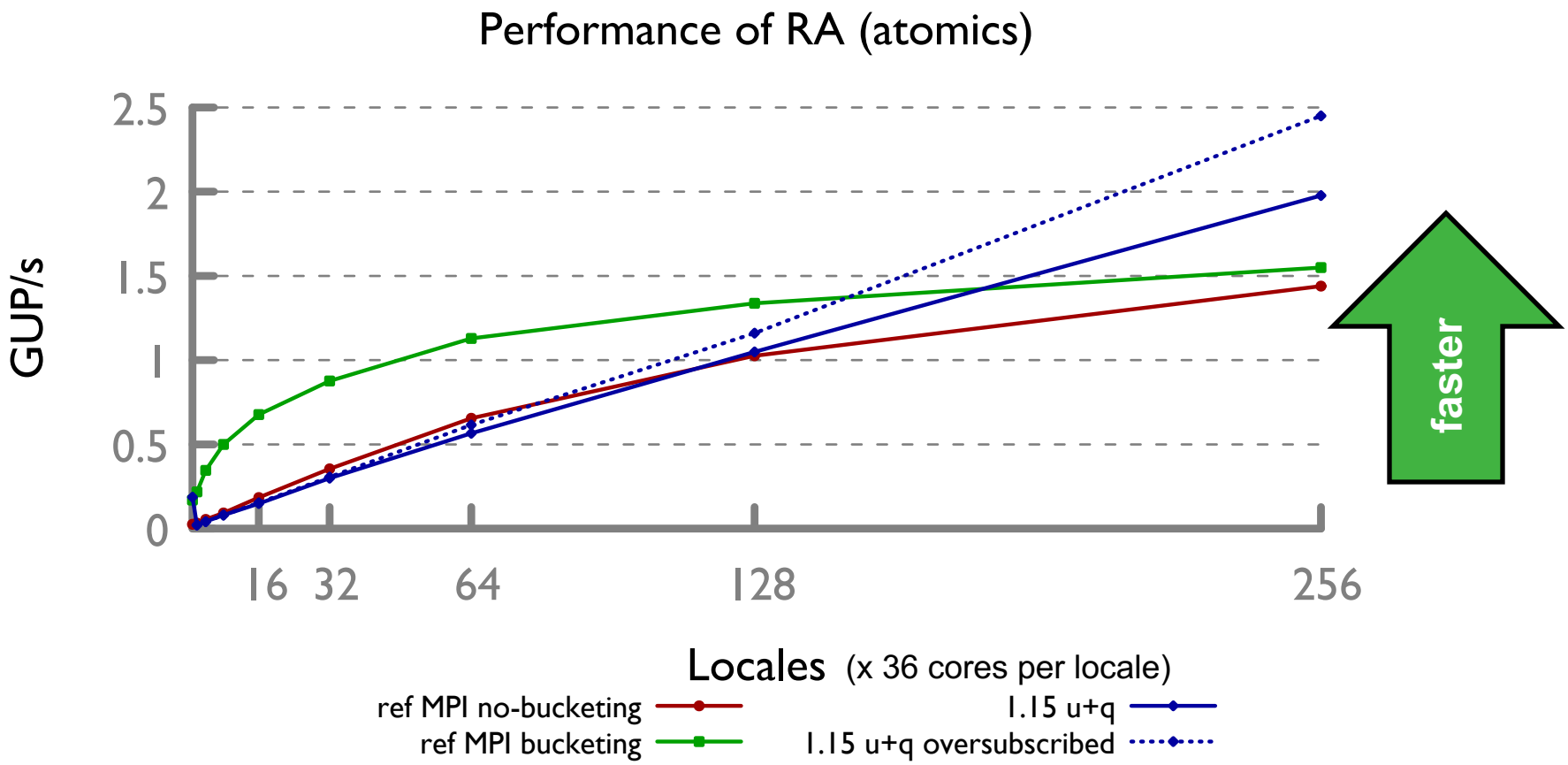
General Chapel Performance Snapshots

LCALS Timings: Chapel 1.15 vs. C [+ OpenMP]

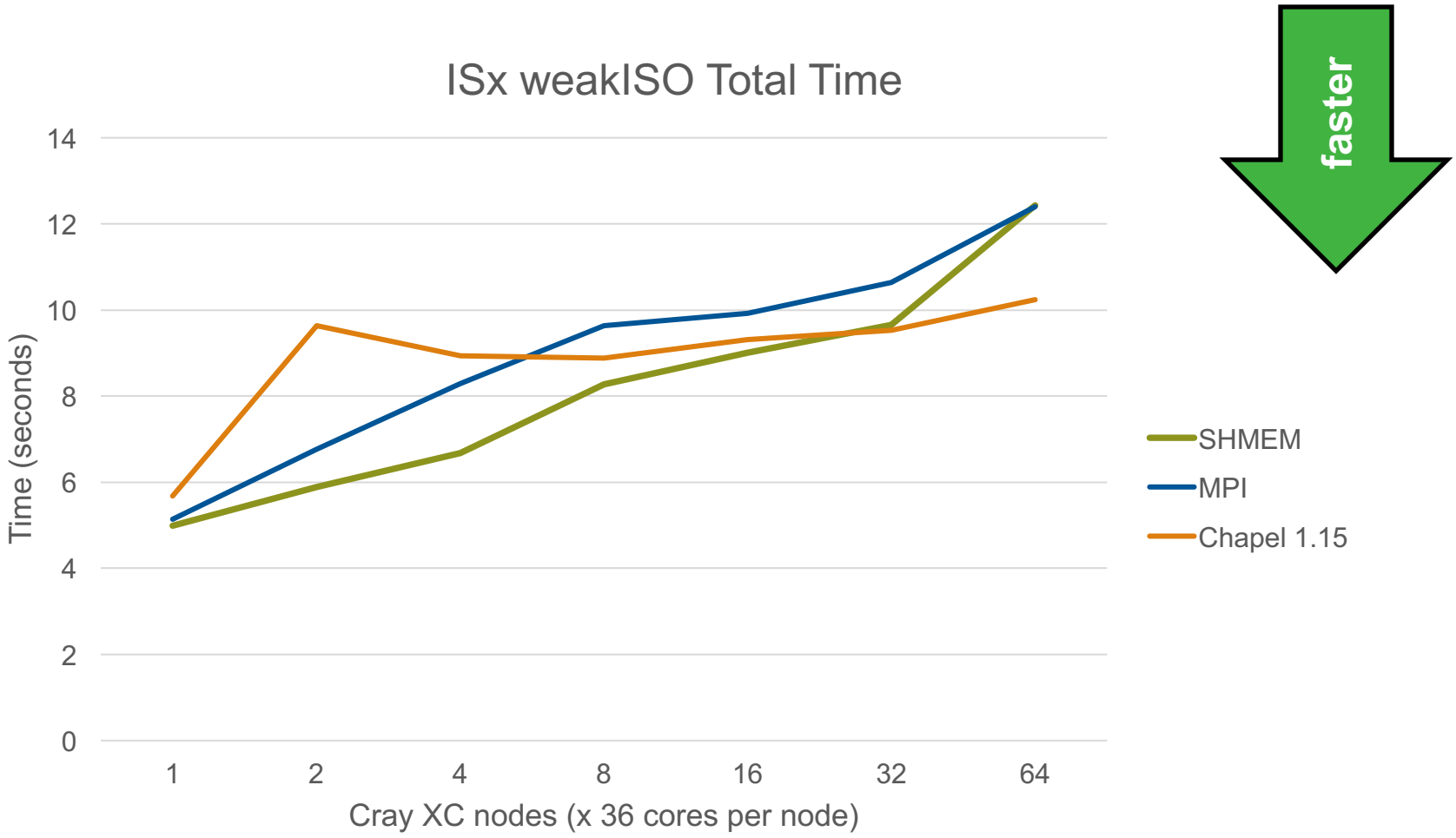
Shared memory performance competitive with hand-coded



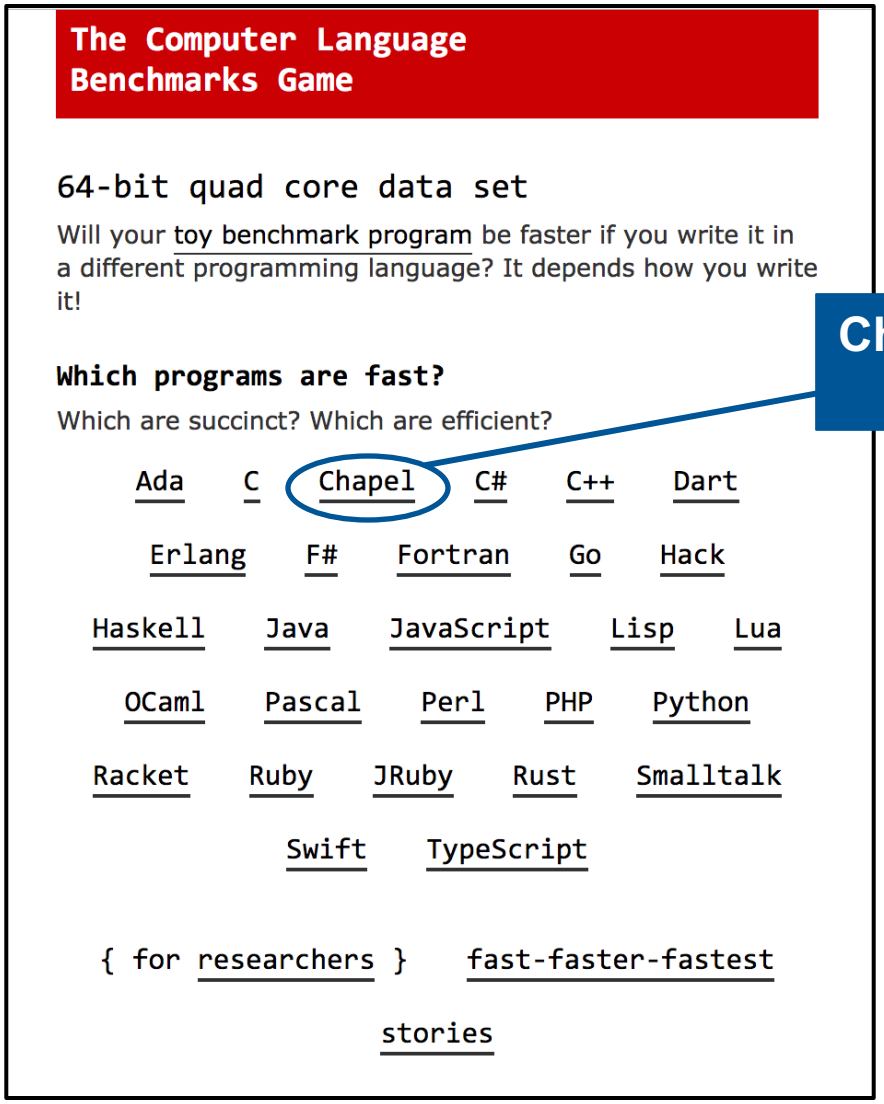
HPCC RA Performance: Chapel 1.15 vs. MPI



ISx Timings: Chapel 1.15 vs. MPI, SHMEM

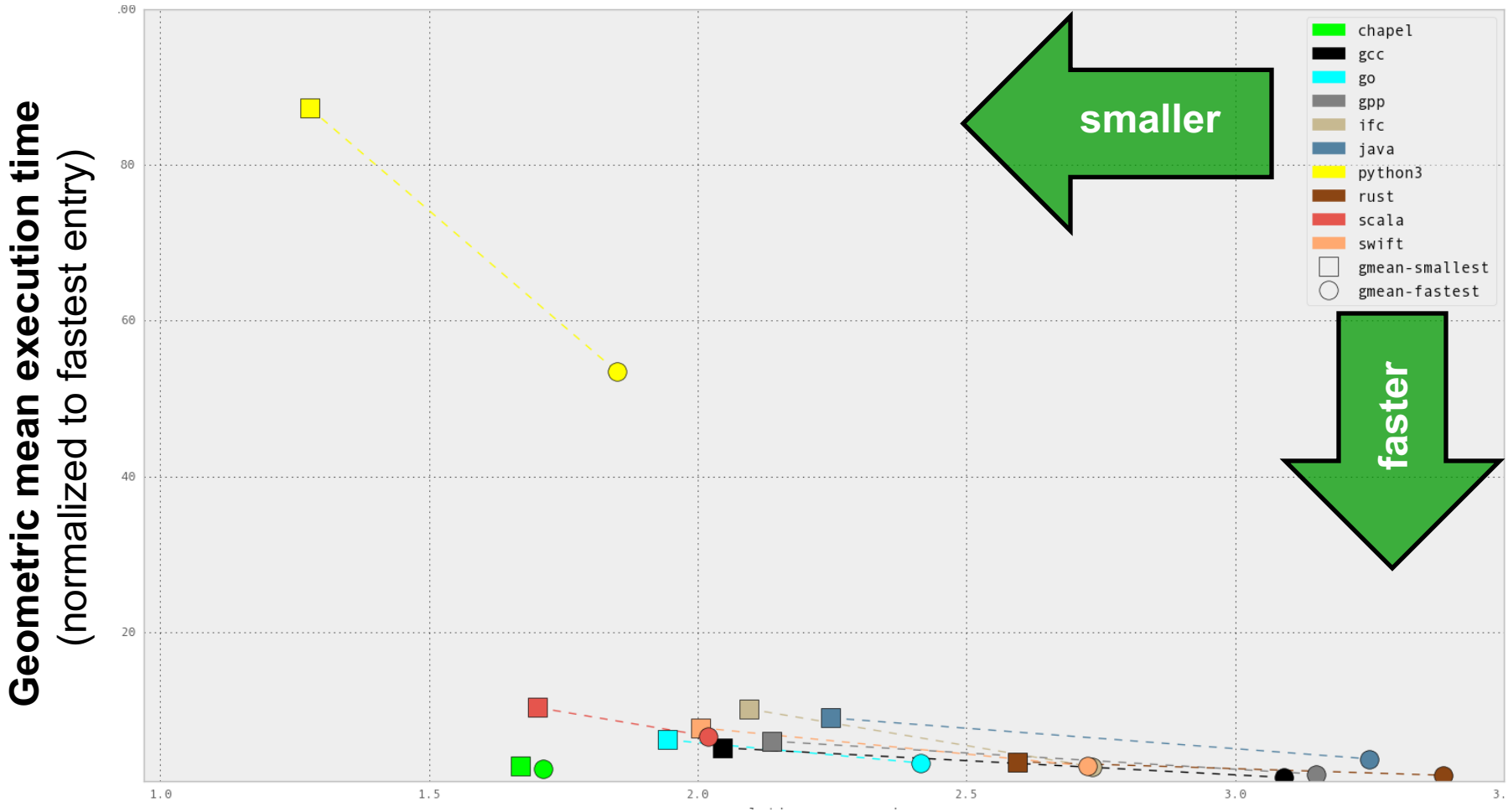


The Computer Language Benchmarks Game (CLBG)

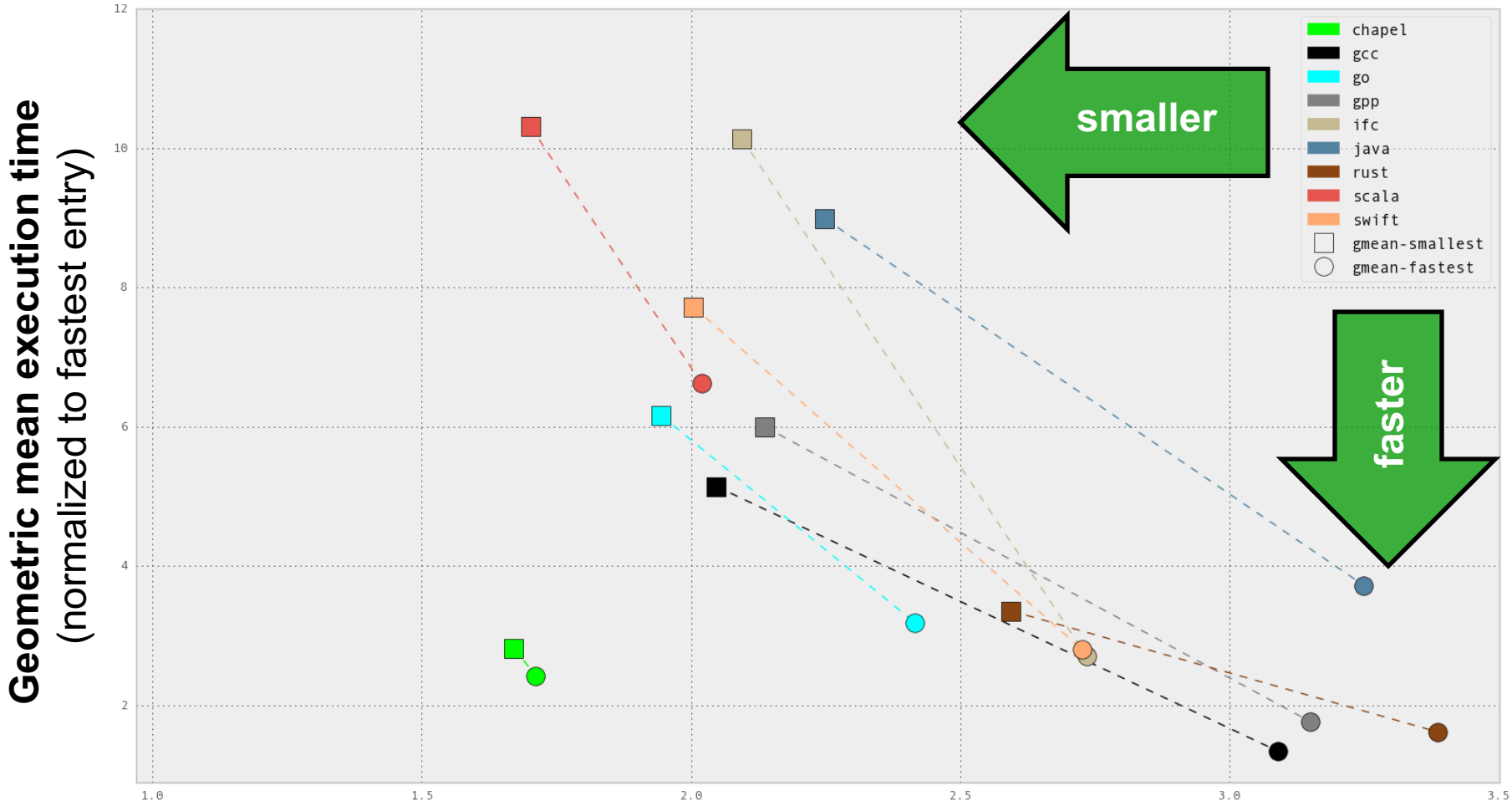


Chapel entry accepted Fall 2016

CLBG Language Cross-Language Summary (May 2017 standings)



CLBG Language Cross-Language Summary (May 2017 standings, no Python)



A Closing Quote
(source: Jonathan Dursi's CHIUW 2017 keynote)

CHIUW 2017 keynote (excerpt)

"My opinion as an outsider…is that Chapel is important, Chapel is mature, and Chapel is just getting started.

"If the scientific community is going to have frameworks for solving scientific problems that are actually designed for our problems, they're going to come from a project like Chapel.

"And the thing about Chapel is that the set of all things that are 'projects like Chapel' is 'Chapel.'"

–Jonathan Dursi

Chapel's Home in the New Landscape of Scientific Frameworks

(and what it can learn from the neighbours)

CHIUW 2017 keynote

Questions?

Chapel Resources

Ways to Track Chapel Remotely

Facebook: http://facebook.com/ChapelLanguage

Twitter: http://twitter.com/ChapelLanguage

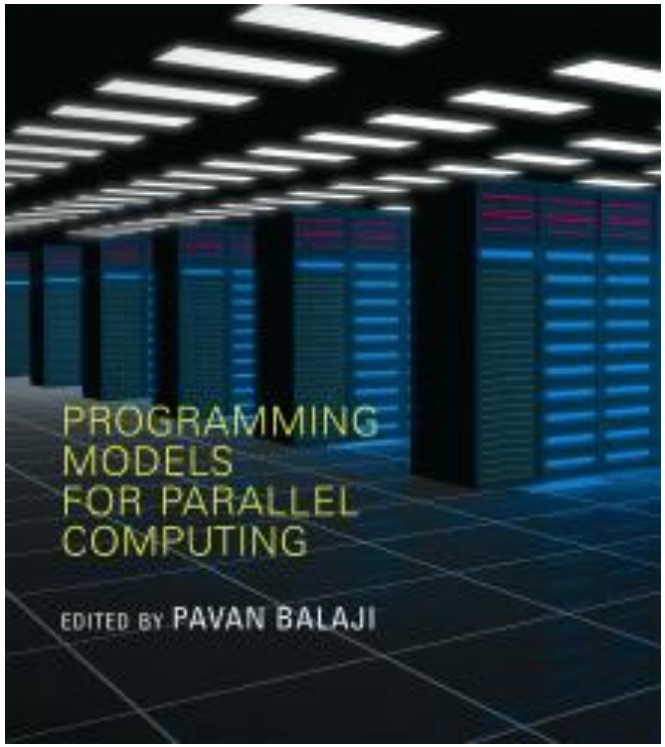
Youtube: https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/

e-mail: chapel-announce@lists.sourceforge.net

Suggested Reading

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 **http://chapel.cray.com/papers.html**

Suggested Short Reads (Blog Articles)

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.cray.com "blog articles" 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
#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

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