From Engineering To Community: Challenges for HPC Programming Language Adoption

Daniel Fedorin, HPE

March 16, 2026

Outline

- Introduction to Chapel
- Community Building
- Community Building for HPC
- Our Experience
  - Stability
  - Developer Tooling
  - Learning Resources

Introduction to Chapel

Chapel is a programming language designed from the ground up with two major philosophical goals, which makes it uniquely suited for writing parallel code

— **Parallelism-by-default**
While other languages provide parallelism as an extension on top of the language, or a third-party library, Chapel keeps parallelism at the forefront.

— **A multi-resolution philosophy**
Chapel provides high-level, elegant parallel programming constructs, but gives the user more control if these constructs prove insufficient.

In some ways, most of Chapel's features are consequences of these two design goals.

Introduction to Chapel

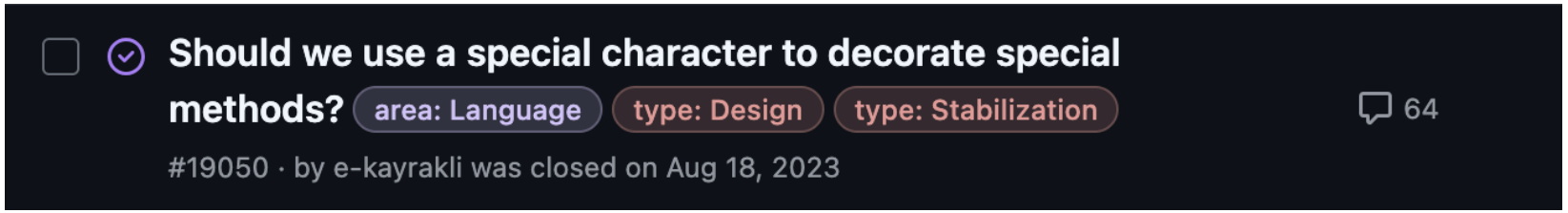
```
use BlockDist;

var Space = blockDist.createDomain(1..n);
var Dst, Src, Inds: [Space] int;
initArrays(Src, Indices);

forall i in Space do
  Dst[i] = Src[Inds[i]];
```

Bale IndexGather (IG)

Community Building

Chapel's Transition to Community-Building

- The 2.0 stabilization push was aimed to address issues with adoption
  - Why would anyone use a language in which every release forces re-writing code?
  - Stabilization efforts took a lot of time, discussion, and engineering



- After the 2.0 stabilization push, the guarantees gave the team confidence to expand our reach

- Now, community-building became an all-hands activity
  - We consider community extremely important

Why Programming Language Community Matters

- Members of the community bring technical expertise
  - This can mean novel problems
  - It can also mean insight into evolve the language

- If you “speak Chapel”, you’d like others to understand you
  - A community provides socialization, support, and feedback
  - Libraries, scaffolds, etc. can accelerate your work

- No one wants a single point of failure
  - a.k.a. the “bus factor”
  - Community is the best way to guarantee resilience

Why [Your Tool] Community Matters

- Members of the community bring technical expertise
  - This can mean novel problems
  - It can also mean insight into evolve the tool

- If you use tool X, you'd like others to understand you
  - A community provides socialization, support, and feedback
  - Libraries, scaffolds, etc. can acce

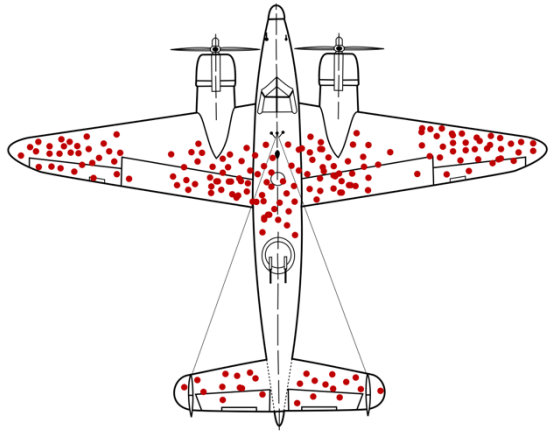
and so on, throughout this talk!

- No one wants a single point of failure
  - a.k.a. the "bus factor"
  - Community is the best way to guarantee resilience

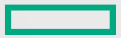
Challenges

Building a community has unique challenges:

- **Measuring Results:** how do you define a “user”? How do you quantify community size?
  - Many possible metrics, but all seem like proxies: likes, views, subscribers, GitHub stars
  - Negative results are typically silence
- **Picking Direction:** a core team has finite time. Where do we spend it?
  - Combined with the previous challenge, hard to tell if efforts are succeeding!
- **Retaining Members:** how do we keep existing community members happy?



Community Building for HPC

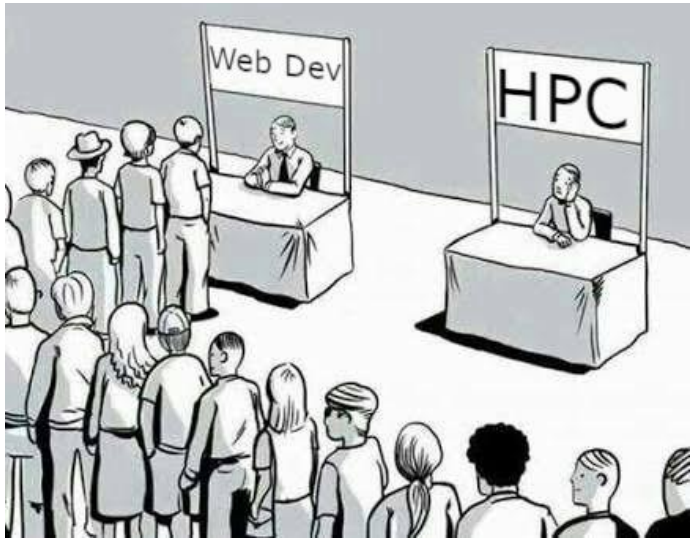


HPC community-building is hard...

- HPC is a comparatively smaller community
  - Not as fun and interactive as, e.g., web development
  - Sometimes requires specialized hardware (cluster? TPU?)
  - Not always included in a core CS curriculum

- HPC is often driven by large, legacy projects
  - Many of people spent a long time to build a firm foundation
  - A new language is hard to sell: “why would I redo all that work?”

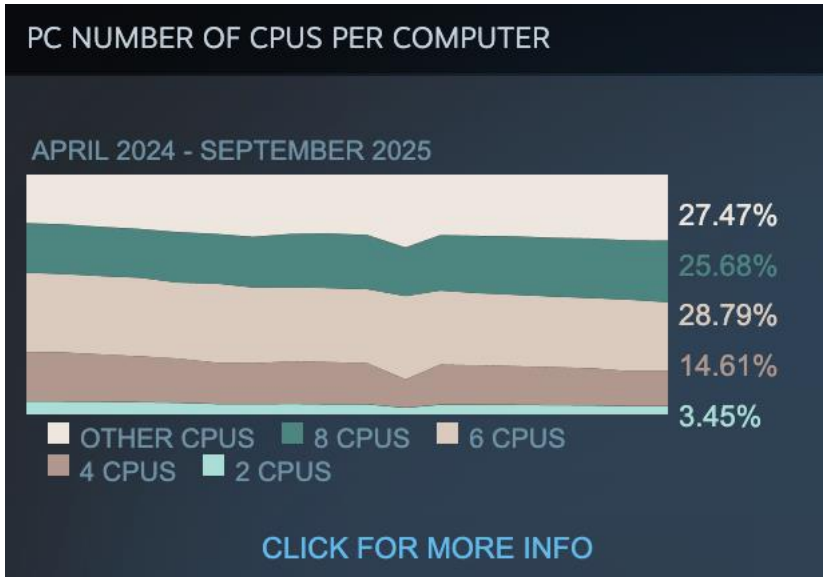
- Flashy results are hard to replicate
  - “I ran on 8192 nodes!” — cool, I’ve got a laptop...
  - In general, performance is relative to your hardware



...but there is much untapped potential

HPC techniques are not unique to supercomputers.

Parallelism and accelerators exist in consumer hardware!



99.97% of surveyed computers have more than 1 CPU!

Measure the Performance of your Gaming GPU with Chapel

Posted on August 27, 2024.

Tags: GPU Programming | How-to | Windows

By: Ahmad Rezaii

If you have a GPU, you have an accelerator!

(and if you have many, you're practically writing distributed code)

...but there is much untapped potential

By broadening the reach of HPC tools, we can help experts and non-experts alike:

- Improved tooling can benefit expert productivity and simplify their work
- Accessible HPC can allow non-experts to use their hardware to its fullest potential

The whole field of HPC can be bigger if we make the right tools available!

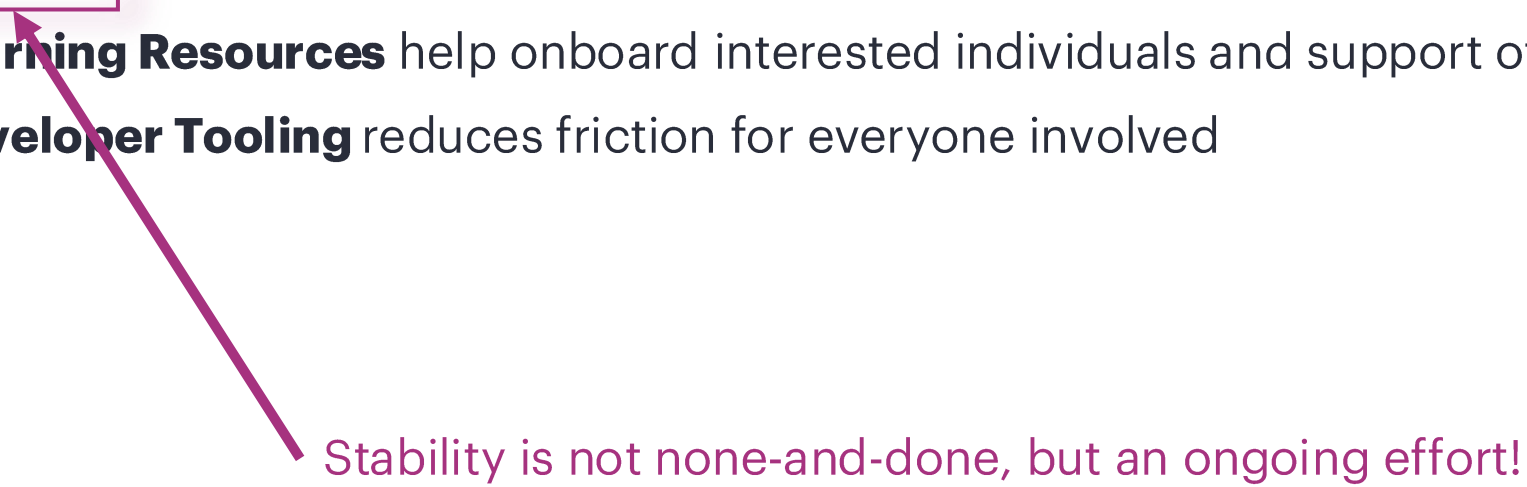
Our Experience

“Behind everything some further thing is found, forever; thus the tree behind the bird, stone beneath soil, the sun behind Urth. Behind our efforts, let there be found our efforts.”
— Gene Wolfe, Citadel of the Autarch

Chapel's Journey Towards Adoption

Since Chapel's 2.0 release, we've found 3 important areas for supporting a growing community:

- **Stability** improves the long-term experience of users and makes future guarantees
- **Learning Resources** help onboard interested individuals and support others' growing expertise
- **Developer Tooling** reduces friction for everyone involved



Stability

Stability is important…

- "Move fast and break things" forces downstream users to keep up
  - Only teams and users with a certain amount of free time can keep up
  - Fighting update-related breakages is just unpleasant
  - Organizations seeking to use Chapel would be discouraged if forced to constantly spend extra resources

- Broken backwards-compatibility invalidates learning resources
  - If you care about maintaining educational material*, you must *also* work to keep up with breaking changes

- If you want to allow large projects in your language, you can't constantly break them
  - I mentioned earlier that these seem to be a relatively common feature of HPC

- AI bonus: knowledge baked into LLMs becomes obsolete
  - Out of the box (without prompts / access to docs) AI will write broken or outdated code

* I think you should care about this! More on that in the next section.

...but there's tension

- Stability readily runs into conflicts with progress
  - Are newly-added features guaranteed to be backwards-compatible in future releases?
  - If existing features are found unsatisfactory, how do we evolve them?
  - Fixing bugs can affect a programming language's semantics, which would change user programs.

- Stability can stifle experimentation
  - If every feature must remain stable, "add it and see" has more cost

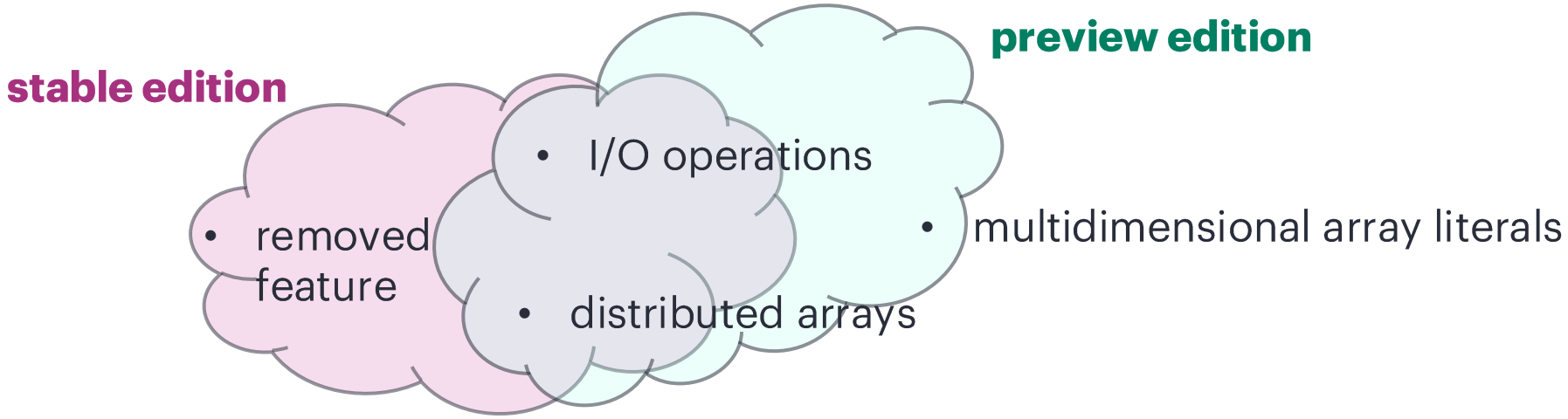
- You should have an established stabilization process
  - Otherwise, you are have a growing, perpetually unstable subset of the language!

"... all observable behaviors of your system will be depended on by somebody."
— Hyrum's Law

Editions

Chapel tackles this using *editions*, inspired by Rust

- Each edition describes a set of language features
  - For the 'stable' edition, these features should remain fully backwards-compatible
  - For the 'preview' edition, these features can appear, disappear, and change.
  - Some unstable features exist outside of an edition and are always unstable.
  - The preview edition eventually 'solidifies' into a stable edition, enabling migration



Learning Resources

Learning Resources

Learning resources answer two distinct, equally important questions,
which help acquire and retain users:

1. How do I do the thing I know I want to do?

Reduces friction from experimenting by answering common questions; e.g., how do I...

- ...read a file?
- ...communicate between nodes?
- ...allocate data on a GPU?

2. What can I do with this?

Users can only try features they know about — so tell them:

- you can define your own parallel iterators!
- you can reason about NUMA domains similarly to reasoning about nodes!
- write generic code over “normal”, distributed, or sparse arrays!

Learning Resources for Everyone

A challenge with learning resources is that virtually no material is universally helpful to all users

- Some users have difficulty reading long, dense documentation
  → might want video tutorials or live, interactive demos

- Some users [myself included] would much rather read!
  → need written learning material, at various levels of density!

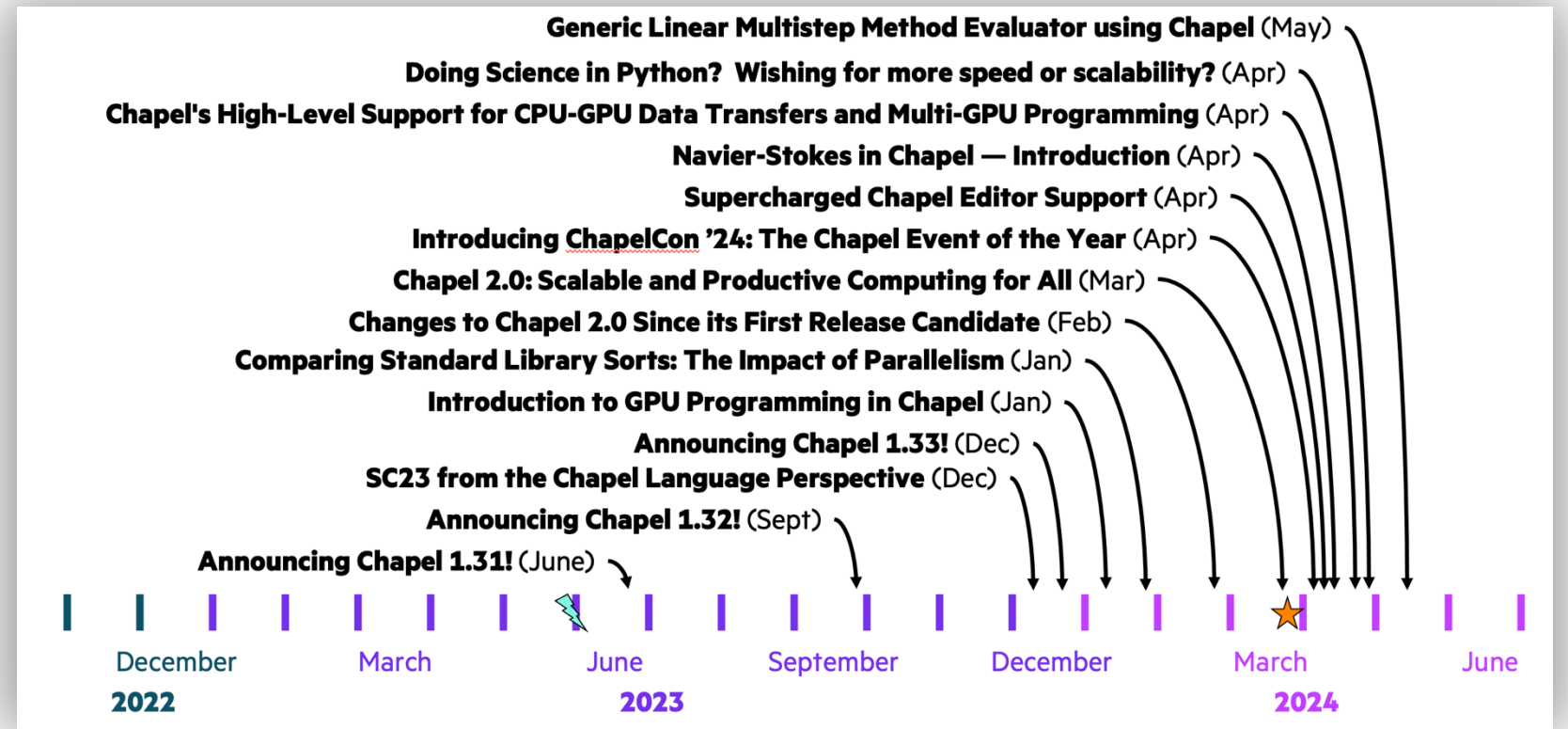
- If trying to access a new segment, you likely need to do a lot of new documentation work
  - E.g., trying to expand beyond “HPC programmers” to “anyone with parallel hardware”

- AI bonus: written knowledge can be consumed by LLMs to train them and augment context

- Open question: given finite time, what do we prioritize?

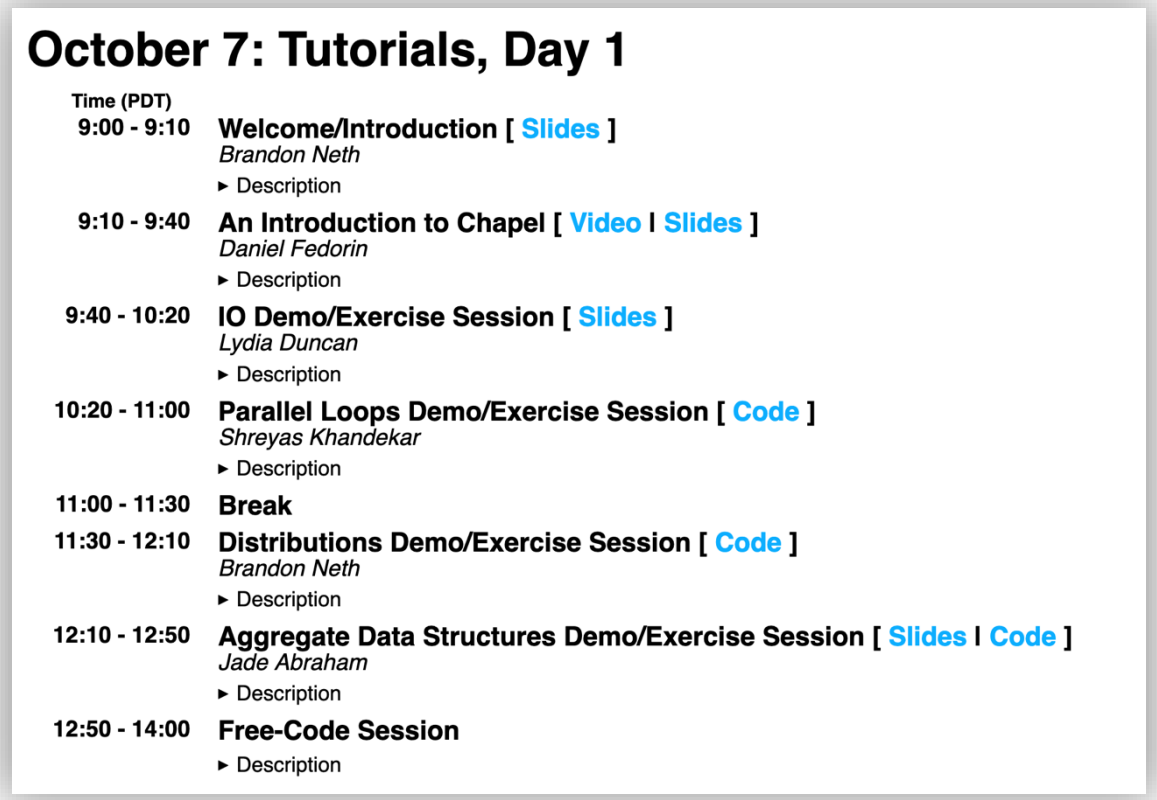
The Blog

- The team had been ramping up the blog prior to the 2.0 release and ramped up posting significantly
- Insofar as interesting content is available, we have done our best to keep this pace up



Tutorials and Presentations

- Adjusted ChapelCon's format to include interactive coding sessions and tutorials
  - In 2024, a dedicated coding day, and a long-form tutorial
  - In 2025, several shorter, practical tutorials for various concepts + coding sessions

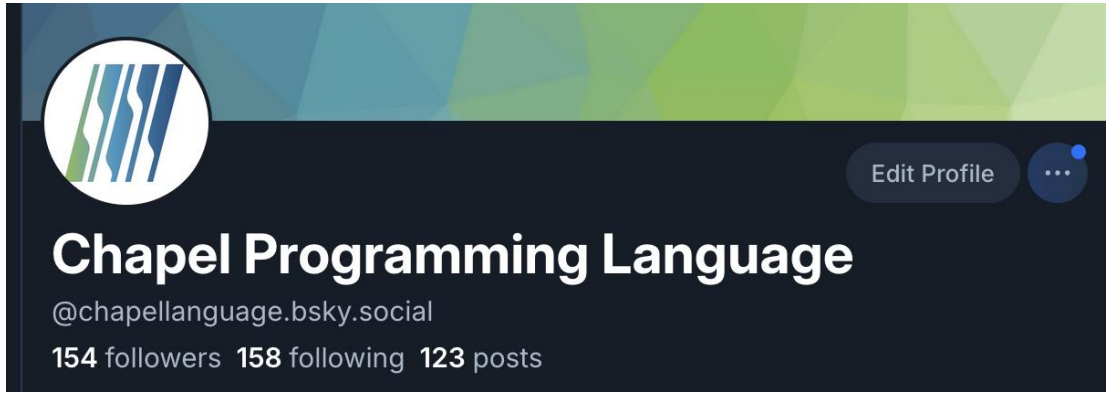


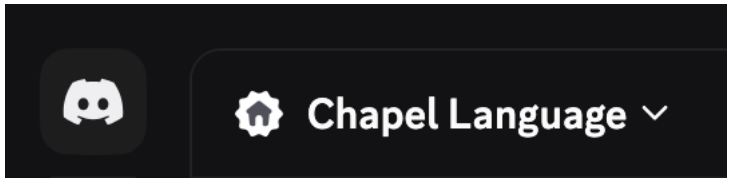
October 7: Tutorials, Day 1

Time (PDT)	
9:00 - 9:10	**Welcome/Introduction [Slides]** *Brandon Neth* ▸ Description
9:10 - 9:40	**An Introduction to Chapel [Video I Slides]** *Daniel Fedorin* ▸ Description
9:40 - 10:20	**IO Demo/Exercise Session [Slides]** *Lydia Duncan* ▸ Description
10:20 - 11:00	**Parallel Loops Demo/Exercise Session [Code]** *Shreyas Khandekar* ▸ Description
11:00 - 11:30	**Break**
11:30 - 12:10	**Distributions Demo/Exercise Session [Code]** *Brandon Neth* ▸ Description
12:10 - 12:50	**Aggregate Data Structures Demo/Exercise Session [Slides I Code]** *Jade Abraham* ▸ Description
12:50 - 14:00	**Free-Code Session** ▸ Description

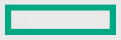
Social Media Presence

- We tried to broaden our presence across social media platforms
- A key concern is not to be exploitative: engage with the platform
  - (but also, engineers can't spend their days reposting memes (I wish 😔))





Developer Tooling



The Need for Tooling

- The more powerful a technology, the more power you need to debug it
- HPC tools are very powerful…
  - Distributed computation
  - Accelerators
  - Parallelism
- …So, when things go wrong, you need to provide a lot of help.

"…debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?"
— Kernighan's Law

The Many Ways to Help

Developer tooling comes in many shapes and sizes!

- Static analysis
  - Catch errors *before* you've spent 30 minutes in a slurm queue!
  - AI bonus: code intelligence can often be used by LLMs to reduce trial-and-error
- Runtime analysis / debugging / profiling
  - Conventional tools (like 'gdb') don't work out-of-the-box in, e.g., a distributed environment!
  - We must expend effort to bring the debugging etc. experience to the level of conventional debugging

Chapel's Developer Tooling Push: Static Analysis

Since 2.0, Chapel has been pushing a language server and linter*.

```
1  use BlockDist;
2  config const n = 100_000,
3               x = 1,
4               y = 3;
5  const D = {1..n, 1..n};
6  var A = Block.createArray(D, int);    Warning: [Deprecation]: 'Block' is deprecated, please use 'blockDist' instead
7  forall a in A {
8    if x < here.id < y {    Error: [NonAssociativeComparison]: comparison operators are not associative
9      a = 1;
10   } else {
11     a = 0;
12   }
13 }
```

src > chpl > linttest.chpl > ...

```
1   var x = true;
2
3   if x == true
4   then {
5     writeln("true");
6   } else {
7     writeln("false");
8   }
9
10
```

*Being a whole new language, I would argue static analysis is easier for us than for a typical project

Chapel's Developer Tooling Push: Debugging

Recently, the team has spent time to enhance the experience of debugging Chapel programs

```
(lldb) c
Process 46026 resuming
Target 0: (debug_real) stopped.
(lldb) on 0
(lldb) f
frame #1: 0x000056219d430f25 debug_real`on_fn_chpl139(myArr=[1..10] int(64)) at debug.chpl:9
   6         breakpoint;
   7         on Locales[0] {
   8           writeln(myArr);
-> 9           breakpoint;
   10        }
   11      }
   12    }
(lldb) p myArr
(_ref(_array(unmanaged [domain(1,int(64),one)] int(64)))) 0x00007f5f4fffcf00 [1..10] int(64): {
  [1] = 1
  [2] = 2
  [3] = 3
  [4] = 4
  [5] = 5
  [6] = 6
  [7] = 7
  [8] = 8
  [9] = 9
  [10] = 10
}
(lldb)
```

We'll Talk More About Tools!

Chapel Tools: Where We Are and Where We Are Going - Jade Abraham, HPE

Wednesday March 18, 2026 3:50pm - 4:15pm CDT

Mississippi River 2

The last few years have seen an explosion of new tools that greatly accelerate Chapel development. This talk will go through some of the new tools Chapel has, new tools we are working on, and what to look forward to in the future.

Conclusion

Open Questions

How do we, as folks in the field of HPC...

- ... best reach an audience of users who are not traditionally interested in HPC?
- ... best facilitate cross-pollination between project communities?
- ... make exciting HPC results accessible to folks with diverse computing resources / hardware?

Thank You