Paying Down Tech Debt: The Challenges of Maintaining Legacy Code in HPC

Jade Abraham (@jabraham17), Advanced Programming Team, HPE

HPSFCon 2026
March 16, 2026

Who am I?

- Software Engineer at HPE working on Chapel
  - https://github.com/jabraham17/
  - https://www.linkedin.com/in/jabraham17

- I work on and help maintain
  - ...the compiler
  - ...the runtime
  - ...the launchers
  - ...the libraries
  - ...the tools
  - ...the CI infrastructure
  - ...

Maintenance is something we can all relate to

Article 28 January 2023

Exploring the Intersection between Software Maintenance and Machine Learning—A Systematic Ma

Oscar Ancán Bastías*, Jaime Díaz and Julio López Fenner

Software Maintenance as Materialization of Common Knowledge

Mace Ojala

Marisa Leavitt Cohn

HTML PDF

PUBLISHED

DOI: https://doi.org/10.17351/ests2023.1325

22 Apr 2024

PeerJ Comput Sci. 2024 Aug 30;10:e2228. doi: 10.7717/peerj-cs

Towards an automated classification phase in the software maintenance process using decision tree

Sahar Alturki [1,✉], Sarah Almoaiqel [1]

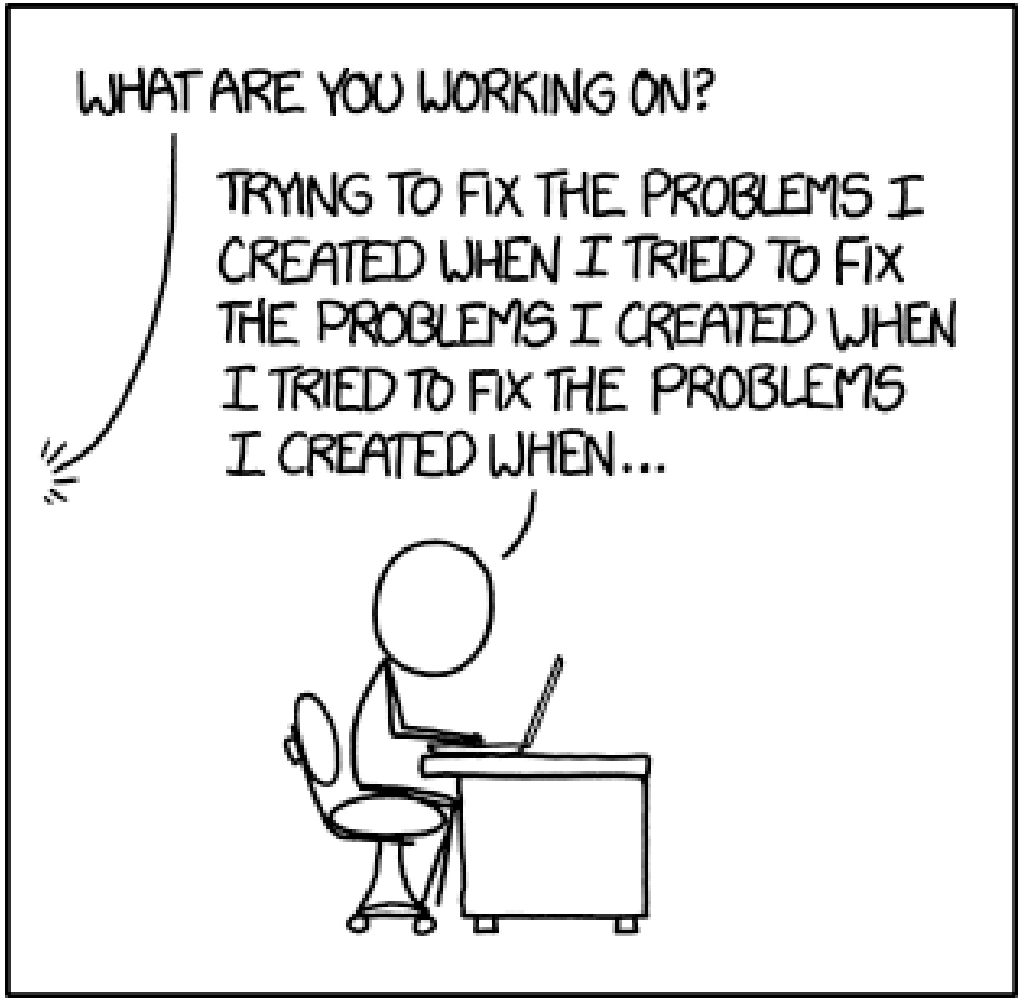
[Submitted on 4 Jul 2024]

Contemporary Software Modernization: Perspectives and Challenges to Deal with Legacy Systems

Marchezan, Alexander Egyed, Rudolf

...

This is a well-explored topic, yet an **unsolved** problem in software engineering



https://xkcd.com/1739/

Don’t even get me started on AI....

This problem only gets worse if we let AI write all our code

Speed at the Cost of Quality

How Cursor AI Increases Short-Term Velocity and Long-Term Complexity in Open-Source Projects

Hao He
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Courtney Miller
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Shyam Agarwal
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Christian Kästner
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Bogdan Vasilescu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

And no, throwing more AI at the problem isn’t the solution either

HPC woes

Beyond the normal software maintenance challenges, HPC has its own quirks

- Decades-old models written in old languages that still need to run fast on constantly evolving architectures
- Hardware changes rapidly
  - For most programmers, a CPU is all there is
  - HPC programmers also worry about NUMA effects, CPU affinity, GPUs, and the network
- Many of the people writing HPC code aren't software engineers
- By its very nature, in HPC, performance trumps all and everything else suffers

Testing is hard

- Hardware matters, but is hard to access
- HPC systems can be quite unique

The answer of course, is CHAPEL

Chapel

```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

SHMEM (Exstack version)

```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
  i0 = i;
  while(i < l_num_req) {
    l_indx = pckindx[i] >> 16;
    pe  = pckindx[i] & 0xffff;
    if(!exstack_push(ex, &l_indx, pe))
      break;
    i++;
  }

  exstack_exchange(ex);

  while(exstack_pop(ex, &idx , &fromth)) {
    idx  = ltable[idx];
    exstack_push(ex, &idx, fromth);
  }
  lgp_barrier();
  exstack_exchange(ex);

  for(j=i0; j<i; j++) {
    fromth = pckindx[j] & 0xffff;
    exstack_pop_thread(ex, &idx, (uint64_t)fromth);
    tgt[j] = idx;
  }
  lgp_barrier();
}
```

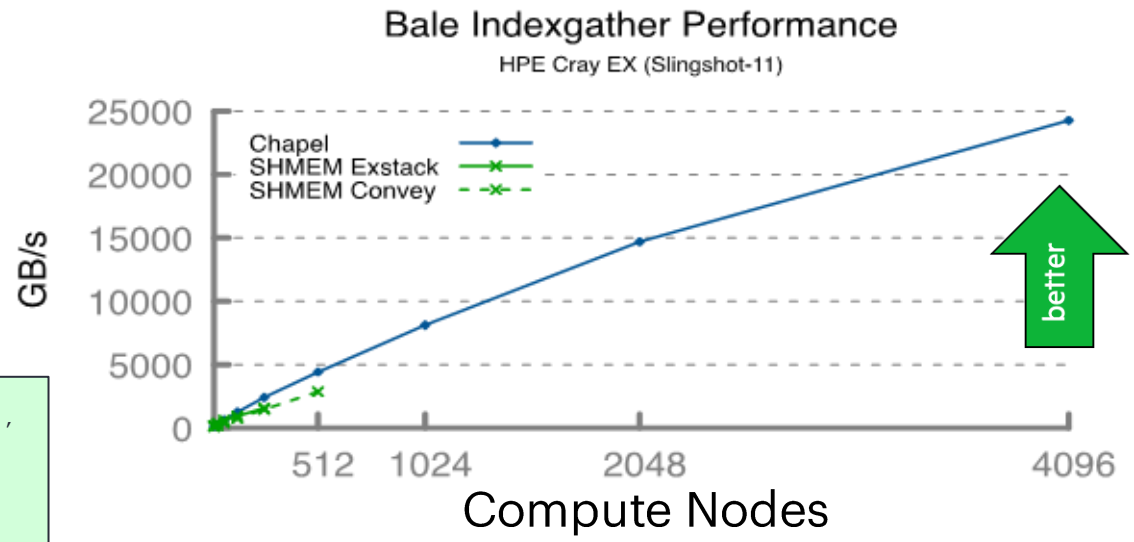
SHMEM (Conveyors version)

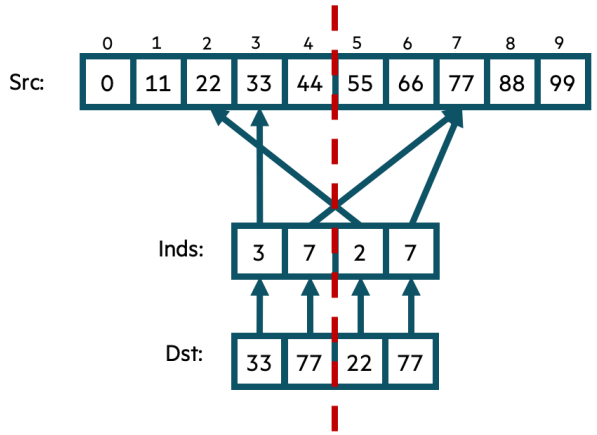
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  for (; i < l_num_req; i++) {
    pkg.idx = i;
    pkg.val = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if (! convey_push(requests, &pkg, pe))
      break;
  }

  while (convey_pull(requests, ptr, &from) == convey_OK) {
    pkg.idx = ptr->idx;
    pkg.val = ltable[ptr->val];
    if (! convey_push(replies, &pkg, from)) {
      convey_unpull(requests);
      break;
    }
  }

  while (convey_pull(replies, ptr, NULL) == convey_OK)
    tgt[ptr->idx] = ptr->val;
}
```





The answer of course, is CHAPEL

Key Features

- A modern language that keeps up with modern hardware
- Strong portability, abstracting away the need for a real HPC system during development
- Get out of the way of science, let scientists express their computations succinctly

I'm a Chapel developer – it's my job to tell you Chapel will solve world hunger 

Sadly, we live in the real world

Challenges for Chapel

New language adoption is hard

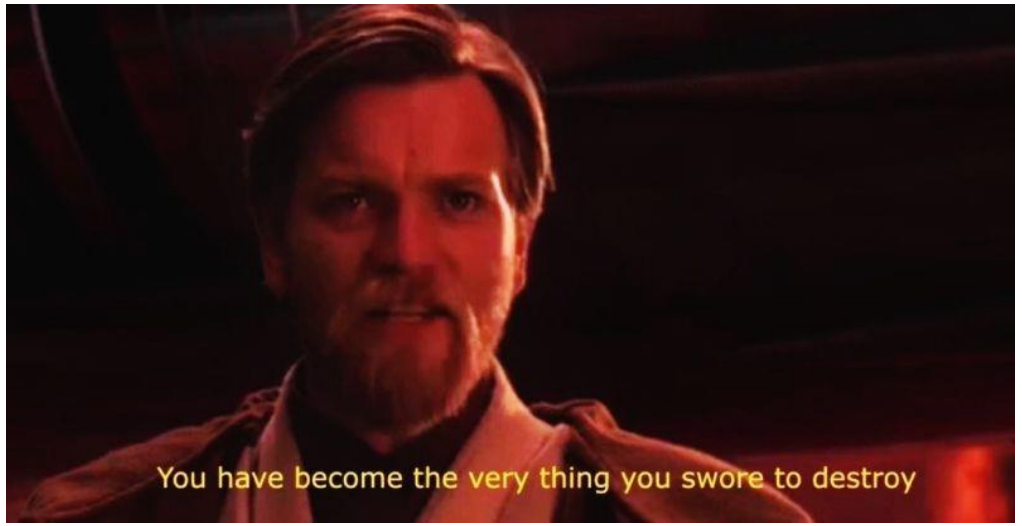
- Significant barriers to entry
- Legacy code wouldn't be a problem if it was easy to rewrite

Writing your code in a different language doesn't automatically make it better

- We need to stop lying to ourselves
- There is no magic bullet

The problems just shift to Chapel's implementation

- Like any good abstraction, the burden of complexity is moved from user to framework
- The abstractions maintainers shoulder the burden for users



These problems don't happen overnight...

Legacy code doesn't just spawn in your code, it slowly creeps in

- Chapel has organically grown over the last 20+ years. We still rely on code that hasn't been touched since the beginning of the project
- A large general-purpose tool, like Chapel, has:
  - a lot of code
  - a lot of dependencies
  - a lot of complexity

New requirements require big changes

- Chapel was designed in a very different HPC world (e.g., no multi-core, no GPUs, different networks)
  - The language has stayed the same for users
  - The implementation has required many changes by maintainers

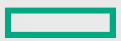
These problems also don't solve themselves

- You must design for the future and refactor
  - e.g., Linux 6.x looks very different from 1.0

But refactoring is scary and hard

- How do I validate my changes?
- How do I show progress if I am just rewriting the existing code?
- How do I know I am rewriting the right things?
- It takes a lot more time to write good, maintainable code than to generate slop

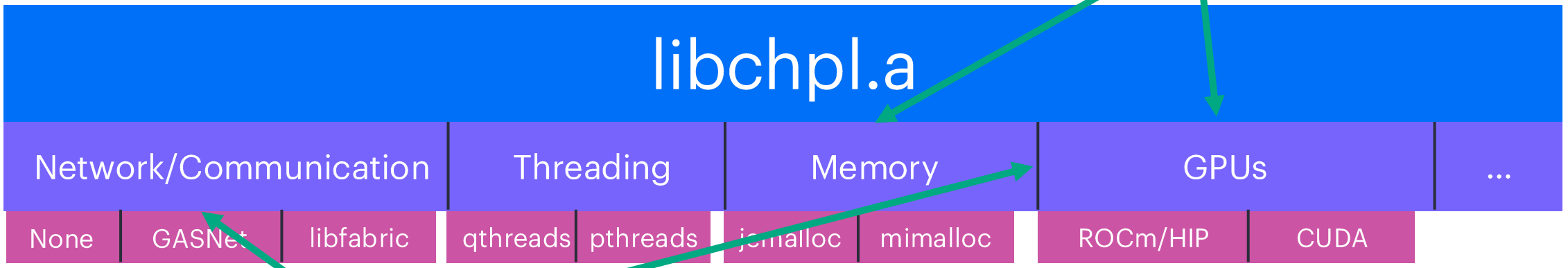
Case Studies in Chapel



The Chapel runtime

At every layer, there is code to be maintained
(and that code's dependencies)

The abstractions in one component are
different from all the others
(you need to be an expert in everything)



Special hardware is required for testing

Testing all configurations results in
combinatorial explosion

"Why does this exist?"
"I don't know but don't touch it"

Launcher Headaches

- Chapel abstracts away system launchers for users
  - Users dislike switching between systems, just let us handle it
  - No more magic slurm/pbs/… incantations, just `./runMyApp`
  - If only it were that easy

- Every HPC center can have its own customizations

- Chapel must balance user experience and maintenance
  - We either specialize for every individual system
  - Or we could work with the broad defaults
    - If you have a custom install, you are just out of luck
    - A long-term wish has been to let users write custom wrappers
  - Currently, we are somewhere in between

Dependencies create pain

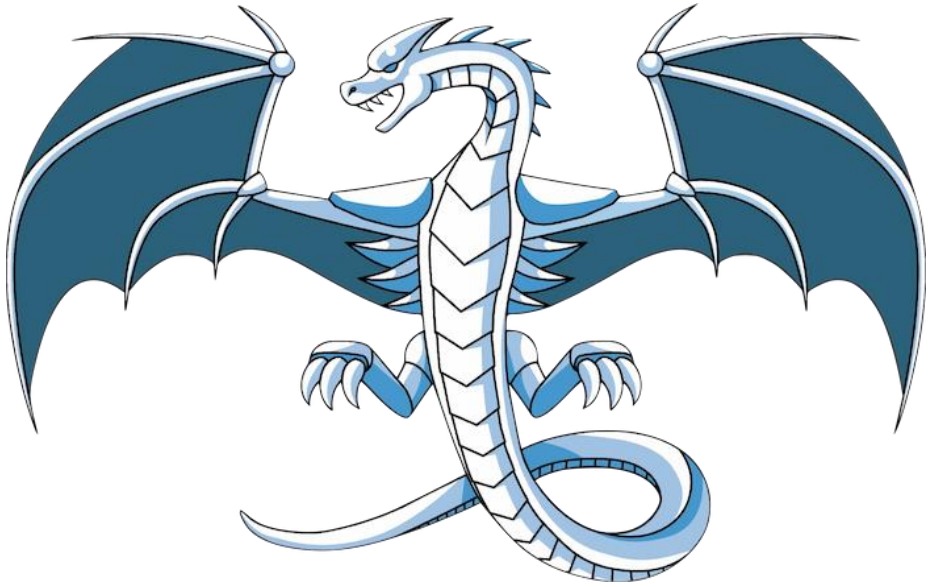
Dependencies are amazing!

- Reusing software is a cornerstone of open-source
- Saves time and work for new things
  - Someone else creates an amazing tool
  - I interact with it through an agreed upon interface (i.e., an API)
  - I can focus on creating something new, someone else focuses on maintaining what already exists

It's just a different kind of legacy code and maintenance

- Am I stuck on some ancient version?
  - The new version broke something I rely on
  - The new version doesn't work with some other dependency
- Someone breaks the semantic versioning contract
- There's a reason it's called "Dependency Hell"

A love story with The Dragon



Chapel relies on LLVM to do optimization and code generation

- Our GPU support is made much easier by LLVM

Chapel supports 8 different versions of LLVM

- Based on the versions available in currently supported OSes
- A new version comes out every ~six months
  - API changes too – every. six. months!
- Each version requires careful porting, correctness testing, performance testing, CI updates, etc

```
#if LLVM_VERSION_MAJOR < 21
      // TargetIntrinsics were removed in LLVM 21
      // we just access them like normal intrinsics
      const llvm::TargetIntrinsicInfo *TII = info->targetMachine->getIntrinsicInfo();
#endif
#if LLVM_VERSION_MAJOR >= 20
      llvm::Intrinsic::ID ID = llvm::Intrinsic::lookupIntrinsicID(cname);
#else
      llvm::Intrinsic::ID ID = llvm::Function::lookupIntrinsicID(cname);
#endif
#if LLVM_VERSION_MAJOR < 21
      if (ID == llvm::Intrinsic::not_intrinsic && TII)
        ID = static_cast<llvm::Intrinsic::ID>(TII->lookupName(cname));
#endif
      if (ID == llvm::Intrinsic::not_intrinsic)
        USR_FATAL("Could not find LLVM intrinsic %s", cname);
```

...ROCm

Semantic Versioning Who?

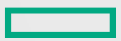
- ROCm changes its behavior on every point release
- This makes for complicated support matrixes
  - Every LLVM version **X** every ROCm minor version

To make things worse...

- ROCm works the best with its own custom LLVM
- Except...AMD LLVM releases don't match LLVM
  - e.g., llvm-amdgpu says it's LLVM 17, but is actually a mix of 16 and 17
- This makes it almost impossible to support upstream LLVM and AMD LLVM

Please don't do this.

Enough Complaining



Solutions?

So far, I've just been complaining. What can we actually do?

- Consistency!
- Teach design, not just code
- Lots of testing
- Make concessions
- Change our mindset
- Engage the community
- ~~Accept our fate~~

Consistency is Key!

Pick an abstraction and stick to it!

- It doesn't have to be perfect, but being consistent lowers the learning curve
- It's great to produce a fancy new way of doing things, but unless the old stuff actually gets phased out, you just create confusion

When the pattern shows its age, it's easier to change

- Eventually the abstraction will have to change
- But if you were consistent from the start, its less work in the long run

It's more than just one project

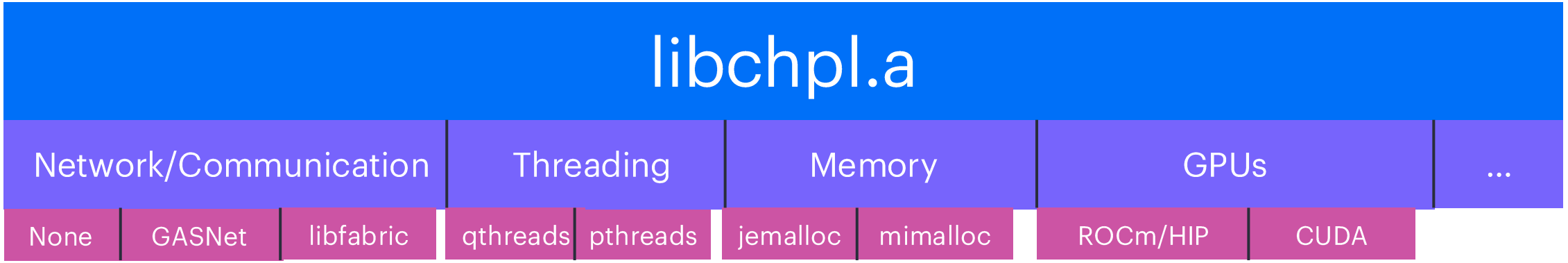
- Can we be consistent across the industry?

• • •

Teach People How to Design

The hard part about programming isn't syntax, its designing solutions to problems

- Every HPC tutorial I've seen is about the mechanics
- Not everyone is a software designer, we need to teach design too
- True engineering is making things work in a sustainable and maintainable way, not just making it work

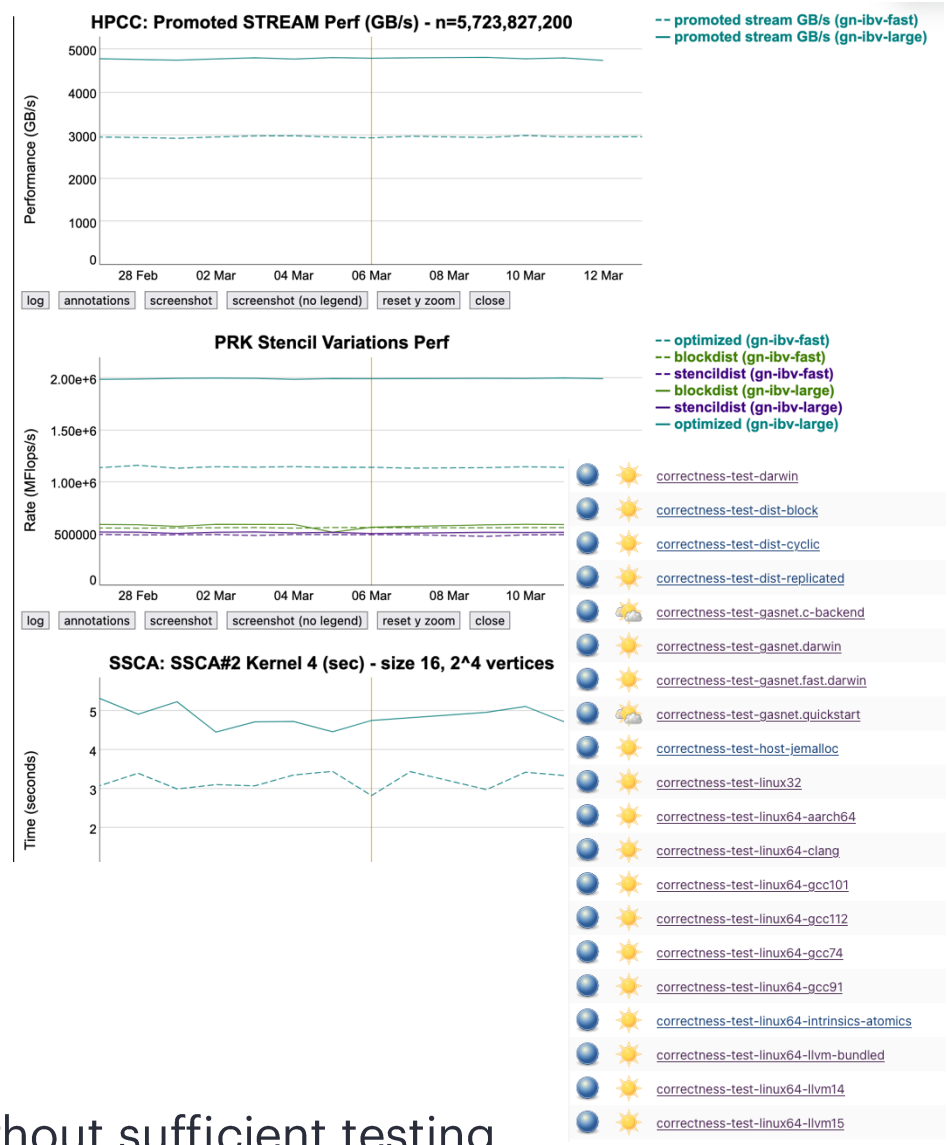


As AI gets better at writing the syntax, this skill becomes even more important

Testing

Good testing lets us make changes fearlessly

- Chapel's testing system keeps us alive
  - 17,000+ tests that run nightly across ~150 different configurations
  - Networks (Slingshot, InfiniBand, Ethernet)
  - Platforms (Linux, MacOS, HPE Cray EX, …)
    - OS distributions (Debian, Fedora, RHEL, …)
  - Compiler flags (--fast, --no-local, --debug)
  - Runtime configurations
  - Performance testing
  - Sanity Checks (leak testing, Asan, Valgrind, --verify)
  - User Applications (Arkouda, CHAMPS, …)
  - …



- The only parts of Chapel I am scared to touch are the parts without sufficient testing
- If I need to go there, the first thing I do is write more tests

Make Concessions

It doesn't have to be perfect the first time

- No matter how perfect you make it, in 10 years someone will rewrite it (hopefully)
- Optimize for clarity and readability, not perfection

Sometimes, we need to be okay with losing a little performance

- I know this is heresy in HPC
- But is that extra performance worth the headache?

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, **and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered**. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%"

— Donald Knuth

Positive Thinking!

Reframing how we think about maintenance

- I don't "have" to work on legacy code, I "get" to work on legacy code
- There is just a really good feeling when you fix a 20-year-old bug

Gamification

Oldest Issue Slayer

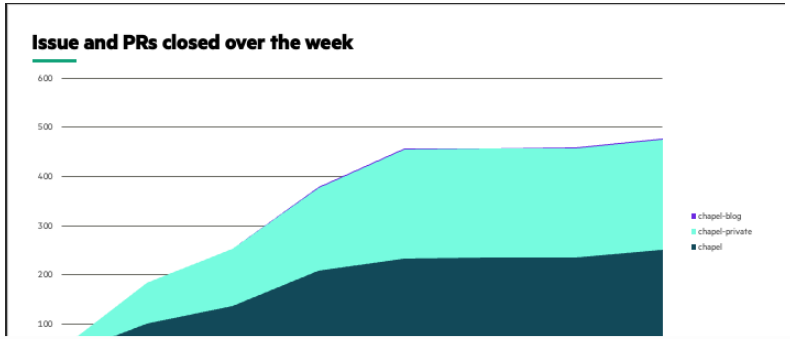
The oldest issue closed during the cleanup week

Rank	Name	Score	Issue
1	jabraham17	7 years, 6 months, 3 days, 3 hours, 13 minutes, 21 seconds	#5095
2	DanilaFe	7 years, 3 months, 29 days, 19 hours, 54 minutes, 45 seconds	#5835
3	lydia-duncan	7 years, 3 months, 26 days, 3 hours, 19 minutes, 16 seconds	#5884
4	benharsh		
5	jeremiah-corrado		
6	ShreyasKhandekar		
7	mppf		

Generous Git Guardian

Most PRs reviewed during the week

Rank	Name	
1	DanilaFe	
2	dlongnecke-cray	
3	mppf	
4	jabraham17	
5	lydia-duncan	
6	vasslitvinov	
7	riftEmber	
8	jeremiah-corrado	
9	e-kayrakli	
10	benharsh	3



Get Help From The Community

We don't have to do this alone!

- The amazing thing about open-source software is people working together
- Can we have a single, low-level network API? (OFI vs UCX vs …)
- Can we have one compiler for all GPUs? (LLVM is so close to this)
- Can we all agree on a good system launcher?

"Good First Issue"

- It's easier to refactor than it is to write new features or fix bugs
- For the latest Chapel release, most of the PRs from new contributors were maintenance related, not new features (Thank You!)

Ways to engage with the Chapel Community

Synchronous Community Events

- Project Meetings, weekly
- Deep Dive / Demo Sessions, weekly timeslot
- ChapelCon (formerly CHIUW), annually

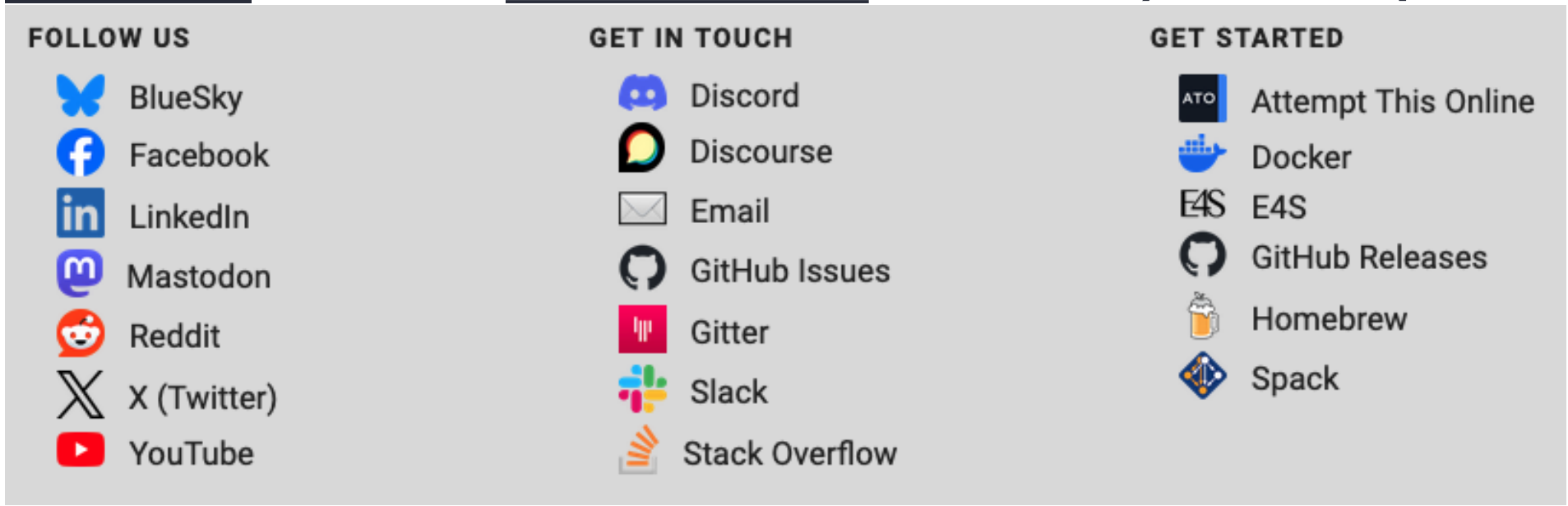
Asynchrounous Communications

- Chapel Blog, typically ~2 articles per month
- Community Newsletter, quarterly
- Announcement Emails, around big events

Social Media

Discussion Forums

Ways to Use Chapel



(from the footer of chapel-lang.org)