Chapel Tools: Where We Are and Where We Are Going

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

HPSFCon 2026
March 18, 2026

Language Tooling is Essential

- Gone are the days where language designers can provide a compiler and that's it
- The 3 big ones
  - Editor Support
  - Static Analysis/Linter
  - Package Manager
- Bonus: Debugger Support

- Users expect tools, a language with no tools also has no users

- Rust is a fantastic example here
  - By its very design, Rust is a tool-first language

Chapel's Smorgasbord of Tools

"Classic" Chapel Tools

- chpldoc
- Mason
- c2chapel
- chplvis
- protoc-gen-chpl
- basic lldb/gdb support
- sanitizer support

"New Age" Chapel Tools

- Dyno/chapel-py
- chpl-language-server
- chplcheck
- chapel-vscode
- lldb support
- chpl-parallel-dbg

Thank you, Dyno!

- Most of these new-age tools are made possible our compiler-as-a-library, Dyno
- Tools don’t reinvent the wheel to understand Chapel code, they use the same logic as the compiler

4:25pm CDT

Evolving a Research Prototype: Dyno, Chapel's New Compiler Front-End - Daniel Fedorin, HPE

MISSISSIPPI RIVER 2

Dyno/chapel-py

- You can’t have an explosion in tools without rapid development

```cpp
void printPreorder(const chpl::uast::AstNode* ast) {
  ast->stringify(std::cout, chpl::DEBUG_DETAIL); std::cout << "\n";
  for (const auto& child : ast->children()) {
    printPreorder(child);
  }
}
int main(int argc, char** argv) {
  chpl::Context context;

  auto& builderResult = chpl::parsing::parseFileToBuilderResultAndCheck(
    &context, chpl::UniqueString::get(&context, std::string(argv[1])), {});

  for (auto i = 0; i < builderResult.numTopLevelExpressions(); i++) {
    printPreorder(builderResult.topLevelExpression(i));
  }

  return 0;
}
```

```python
import chapel
import sys

context = chapel.Context()
top_level_nodes = context.parse(sys.argv[1])

for node in top_level_nodes:
    for n in chapel.preorder(node):
        print(repr(n))
```

- chapel-py provides an easy-to-use Python interface to a powerful and performance C++ library

chpl-language-server

- Provides editor support for Chapel
  - Generic tool that works in any editor that supports LSP

- Key Features
  - Goto-Definition
  - Type resolution
  - Live Error Reporting

- Gives Chapel programmers a modern editor experience and saves time
  - When fixing errors, no context switch between editor and compiler

chpl-language-server

```
Show Generic | Show Instantiation | Show Instantiation
proc binarySearch(arr, target) {
  var low = 0;
  var high = arr.size - 1;
  while low <= high {
    var mid = (low + high) / 2;
    const midVal = arr[mid];
    if midVal < target {
      low = mid + 1;
    } else if target < midVal {
      high = mid - 1;
    } else {
      return mid;
    }
  } while low <= high
  return -1;
} proc binarySearch(arr, target)
```

chpl-language-server: What's Next?

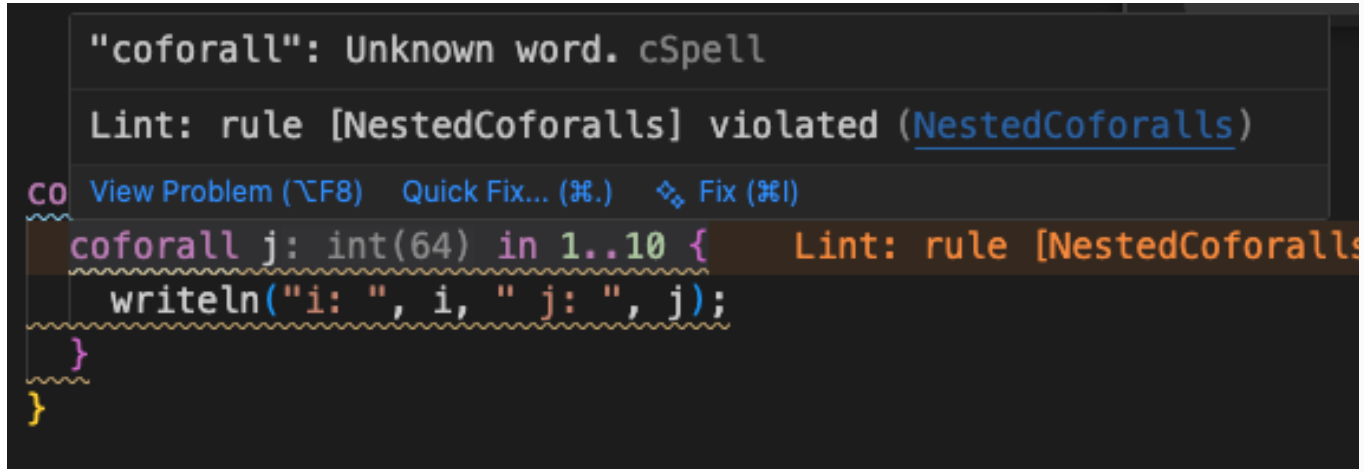
- Type Resolution Improvements
  - The new frontend can resolve most Chapel code, but not all
  - Provide more information for instantiations of generic code

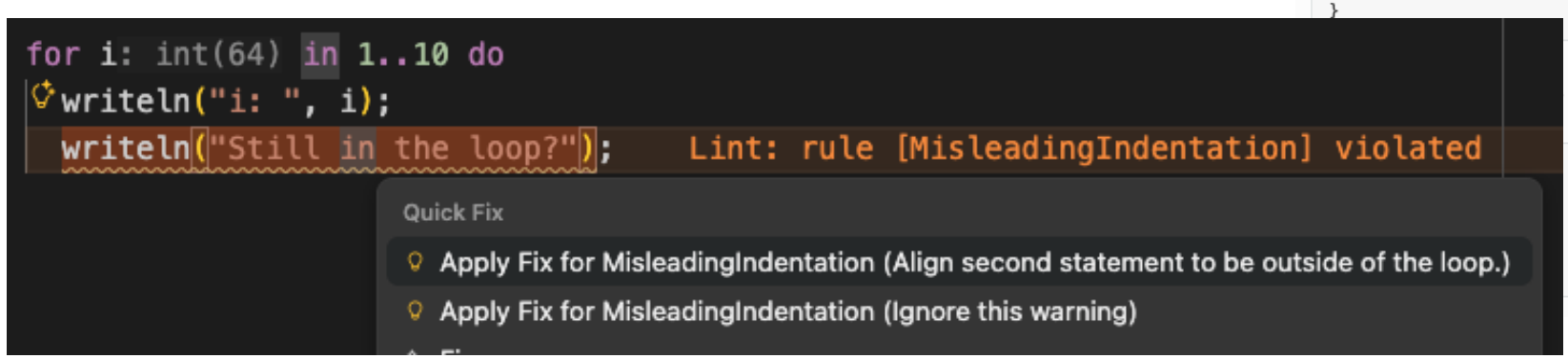
- Better support for large projects
  - Works great on large projects, but some of the more advanced features need some work

- But mostly, chpl-language-server is a fully featured and polished tool

chplcheck

- A linter for Chapel anti-patterns
  - Check for consistent style
  - Find potential bugs





NestedCoforalls

Is enabled by default? No

Warn for nested 'coforall' loops, which could lead to performance hits.

Nested coforall loops can lead to performance issues because each coforall creates one task per iteration. A nested coforall creates N×M tasks, which can overwhelm the runtime with excessive task creation overhead.

This creates 10 × 10 = 100 tasks (one for each (i,j) pair). With larger ranges, this quickly becomes problematic: e.g., 1000 × 1000 = 1,000,000 tasks

```
coforall i in 1..10 {
  coforall j in 1..10 {
    writeln(i, j);
  }
}
```

(one per value of i). The inner loop runs serially within each task.

chplcheck: What's Next?

- More Lints!
  - As we codify a more consistent Chapel style, the linter becomes an even more important tool

- More static analysis
  - The more anti-patterns and potential problems we can highlight for users, the better
  - Look beyond the syntax
    - e.g., can we identify for users where there might be excessive communication and suggest better patterns?

Mason

- Chapel's package manager – inspired by 'cargo'
- An older Chapel tool that has recently seen lots more love and attention

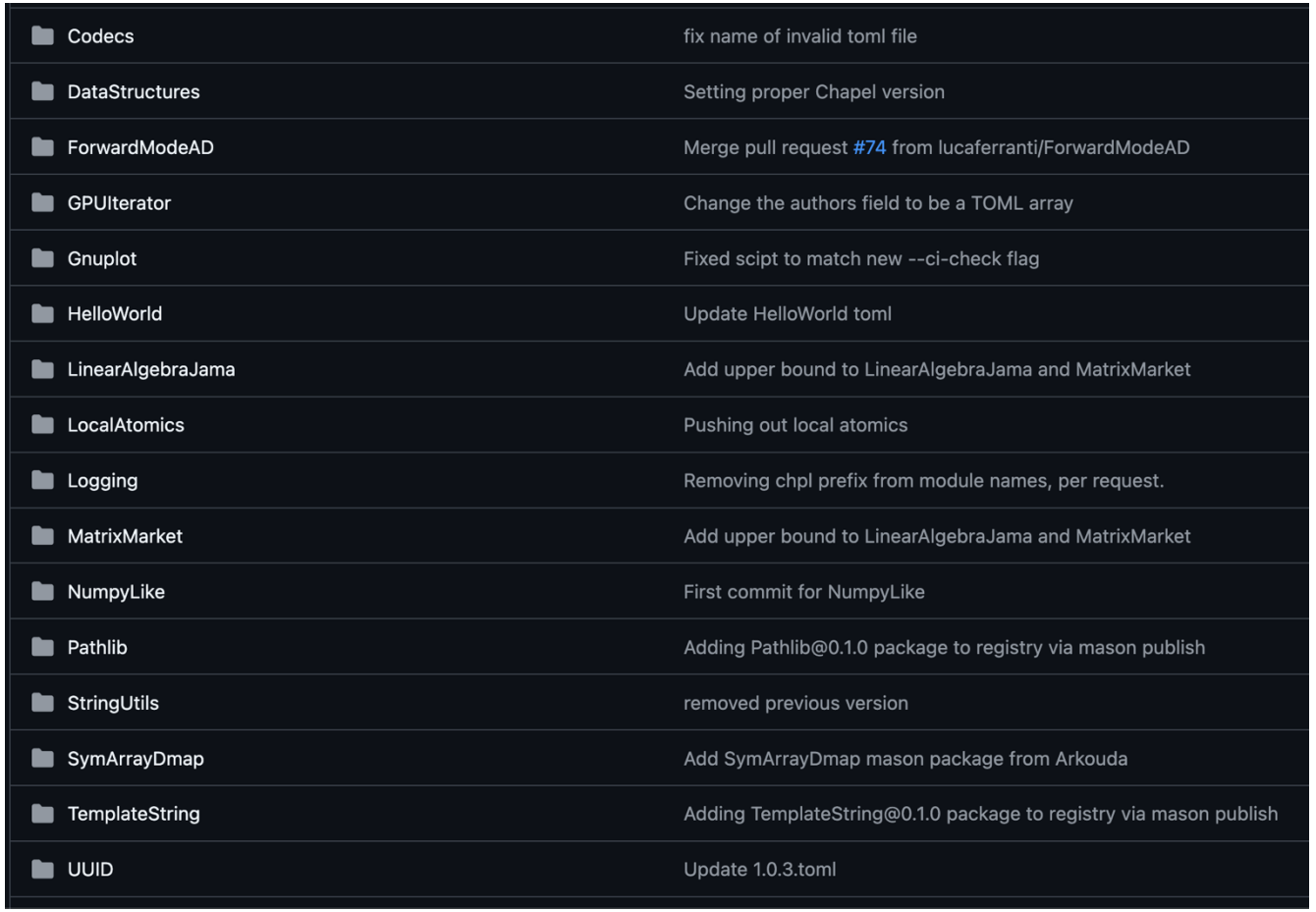
- Improving on the old
  - Major refactors to update the code for "modern Chapel"
  - Writing many new tests

- New features
  - Support for building non-Chapel code
    - Ideal for interop scenarios
  - Better documentation
  - Subtle QOL enhancements
    - e.g., allowing specifying compiler options, rather than just a string

Mason: What's Next?

- We want more Mason packages!

- Better built-in tools for…
  - testing
  - benchmarking
  - debugging



Codecs	fix name of invalid toml file
DataStructures	Setting proper Chapel version
ForwardModeAD	Merge pull request #74 from lucaferranti/ForwardModeAD
GPUIterator	Change the authors field to be a TOML array
Gnuplot	Fixed scipt to match new --ci-check flag
HelloWorld	Update HelloWorld toml
LinearAlgebraJama	Add upper bound to LinearAlgebraJama and MatrixMarket
LocalAtomics	Pushing out local atomics
Logging	Removing chpl prefix from module names, per request.
MatrixMarket	Add upper bound to LinearAlgebraJama and MatrixMarket
NumpyLike	First commit for NumpyLike
Pathlib	Adding Pathlib@0.1.0 package to registry via mason publish
StringUtils	removed previous version
SymArrayDmap	Add SymArrayDmap mason package from Arkouda
TemplateString	Adding TemplateString@0.1.0 package to registry via mason publish
UUID	Update 1.0.3.toml

- Install command line tools, just like 'cargo', 'npm', 'pip'
  - Super-computer powered command line tools?!

chpl-parallel-dbg

- Chapel has long had limited support for debugging in gdb/lldb
  - It worked, but you really needed to be a compiler expert
- We've recently made great steps towards a better experience for both

- But multi-locale debugging still was lacking
  - The best method was to X-forward a bunch of terminal windows, one for each locale/node

- chpl-parallel-dbg provides a much better debugging experience for multi-locale code
  1. Launch your program in debug mode
  2. Attach the parallel debugger
  3. Step through and inspect multi-locale code as if it was just multiple threads on one node

```
use BlockDist;
proc main() {
  var arr = blockDist.createArray(0..#10, int);
  arr = arr.domain;
  coforall loc in Locales do on loc {
    const myVar = loc.id;
    writeln("Hello from locale ", myVar);
    const mySlice = arr[arr.localSubdomain()];
    writeln("My slice is: ", mySlice);
    import Debugger; Debugger.breakpoint;
  }
}
```

Switch execution between nodes

Inspect two different locales in a single session

```
(lldb) on 0
(lldb) f
frame #1: 0x000056186b529db0 example_real`on_fn_chpl197(arr=0x00007fef3edfcf30,
   7         writeln("Hello from locale ", myVar);
   8         const mySlice = arr[arr.localSubdomain()];
   9         writeln("My slice is: ", mySlice);
-> 10        import Debugger; Debugger.breakpoint;
   11      }
   12    }
(lldb) p mySlice
(ChapelArray::[domain(1,int(64),one)] int(64)) [0..4] int(64) {
  [0] = 0
  [1] = 1
  [2] = 2
  [3] = 3
  [4] = 4
}
(lldb) c
Process 56685 resuming
Target 1: (example_real) stopped.
(lldb) on 1
(lldb) f
frame #1: 0x000055919c96ddb0 example_real`on_fn_chpl197(arr=0x00007fa8cc9fcf30,
   7         writeln("Hello from locale ", myVar);
   8         const mySlice = arr[arr.localSubdomain()];
   9         writeln("My slice is: ", mySlice);
-> 10        import Debugger; Debugger.breakpoint;
   11      }
   12    }
(lldb) p mySlice
(ChapelArray::[domain(1,int(64),one)] int(64)) [5..9] int(64) {
  [5] = 5
  [6] = 6
  [7] = 7
  [8] = 8
  [9] = 9
}
Target 0: (example_real) stopped.
(lldb)
```

chpl-parallel-dbg: What's Next?

- Just in general, even better debug information for Chapel programs

- Bulk actions
  - e.g., "on 0,1,2,3 print myVar"

- Can we generalize beyond Chapel? Are other communities interested?

Tooling Wishlist

- Auto-formatter
  - In large projects with multiple developers, being able to enforce consistent formatting via tooling is essential

- Profiler
  - Help users find and fix performance issues
  - Chapel already works with standard tools, can we make those integrations better?

- Language Migration and Interoperability
  - Can I rewrite my legacy Fortran app in Chapel?
  - c2chapel currently generates basic wrappers, can we help library authors automate away their maintenance?

Ways to engage with the Chapel Community

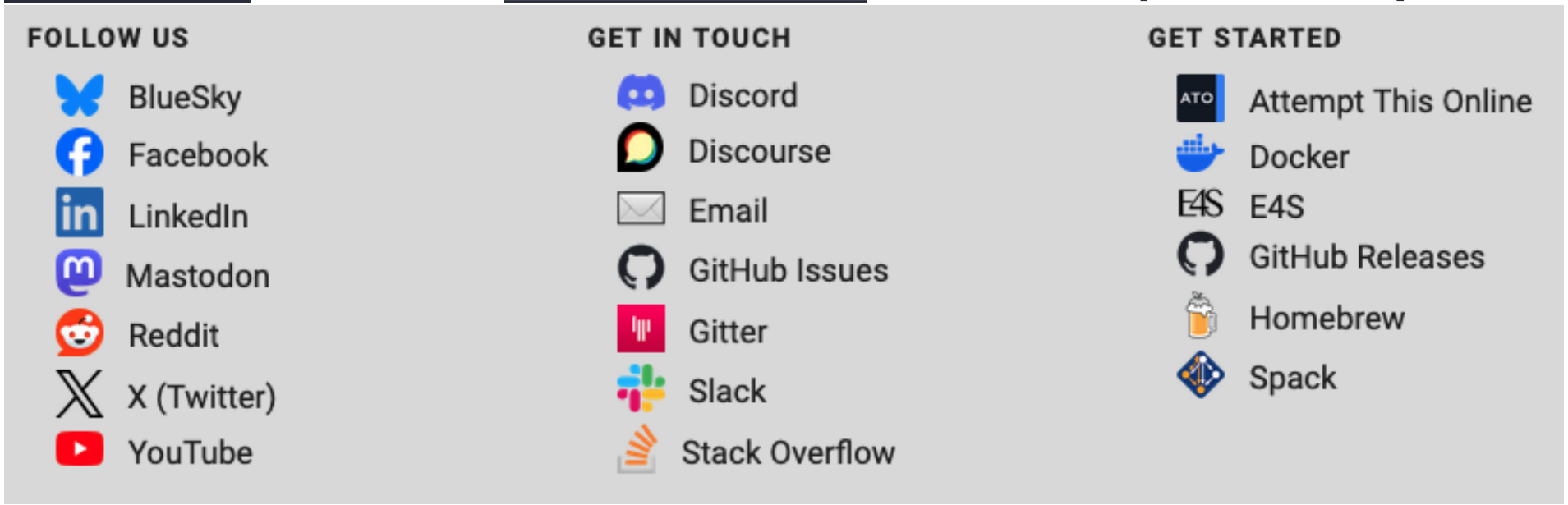
Synchronous Community Events

- 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)