Hewlett Packard Enterprise

CHAPEL 1.25 RELEASE NOTES: HIGHLIGHTS

Chapel Team

September 23, 2021

HIGHLIGHTS OF CHAPEL 1.25

- Chapel 2.0 / Language and Library Highlights
- Performance Improvements and Studies
- Compiler Improvements
- Targeting GPUs with Chapel
- Summary

CHAPEL 2.0 / LANGUAGE AND LIBRARY HIGHLIGHTS

CHAPEL 2.0

Background:

- Over the past few years, we have been working toward a forthcoming Chapel 2.0 release
- Intent: stop making backward-breaking changes to core language and library features
  - thereafter, use semantic versioning to reflect if/when such changes are made

This Release:

- Major language-related fixes have largely wound down
- Primary remaining effort is on stabilizing the standard libraries

LANGUAGE / LIBRARY IMPROVEMENTS

‘foreach’ loops: express parallel loops that should be implemented by the current task

- help indicate opportunities for vectorization or GPU execution when a ‘forall’ loop’s tasks would be overkill

```
foreach i in 1..n do    // assert that this loop is order-independent
  a[i] = b[p[i]];
```

‘manage’ statements: support Python-like context management

- prioritized to support resizable collections of non-nilable classes

operators: prototyped in 1.24, now ready for use

- address an otherwise vague namespace issue

```
operator R.+(x: R, y: R) { … }
```

‘ArgumentParser’ package module: in support of richer command-line options than ‘config’ supports

- developed in support of the ‘mason’ package manager

other improvements: progress with interfaces, refinements to ranges, etc.

- done in support of Chapel 2.0, user feedback and requests

STANDARD LIBRARY STABILIZATION

- This represents the lion's share of the remaining work for Chapel 2.0

Module	1.24	1.25
Builtins	Review Started	Stable
Chapel Env		
Heap		
List		Review Started
Map		Review Started
Set		
CommDiag.	Review Started	Not 2.0
Memory		Not 2.0
FileSystem		
IO		Review Started
Path	Review Started	Stable
Reflection		Review Started
Types		
BigInteger		Review Started
BitOps		Not 2.0
GMP		Not 2.0
Math		
Random		
Barriers		Review Started
DynamicIters		Not 2.0
VectorizingIterator		Not 2.0
CPtr / SysCTypes		Review Started
Spawn		Review Started
Sys		Review Started
SysBasic		Review Started
SysError		
DateTime		
Help		Not 2.0
Regexp	Review Started	Progress
Time	Review Started	Progress
Version		
String / Bytes		Review Started
Ranges / Domains/ Arrays	Review Started	Review Started
Shared / Owned	Review Started	Review Started
Errors		

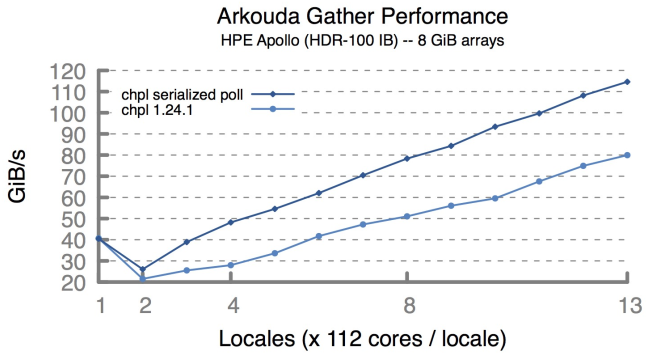
Stable **Progress** **Review Started** **Not 2.0**

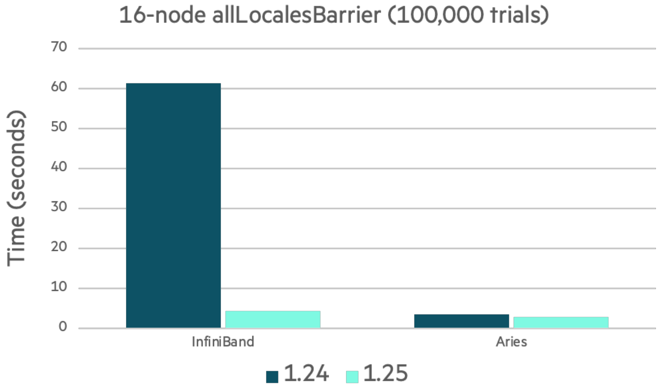
PERFORMANCE IMPROVEMENTS AND STUDIES

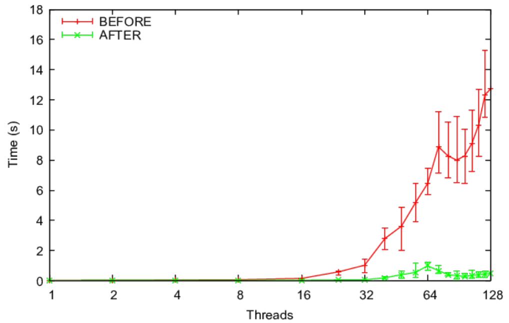
SO MANY PERFORMANCE IMPROVEMENTS

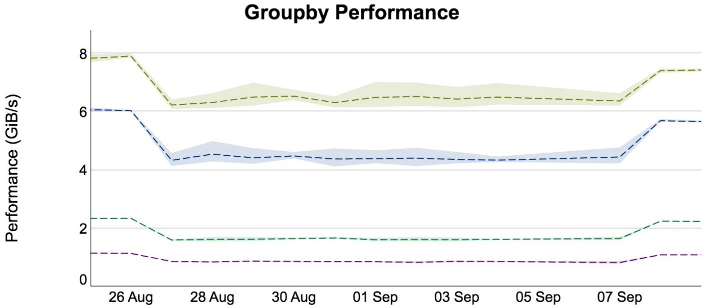
Primarily motivated by…

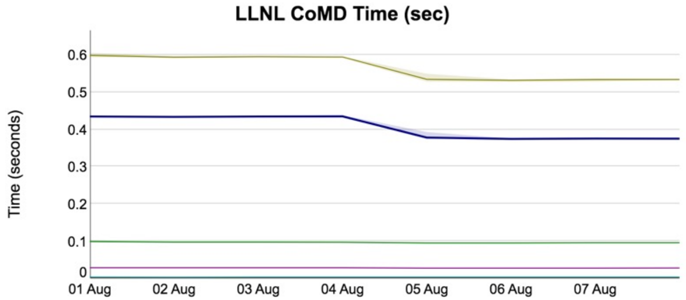
- …user code, especially Arkouda
- …targeting new platforms
  - InfiniBand-based systems
  - high core-count chips like AMD Rome
  - large-memory nodes

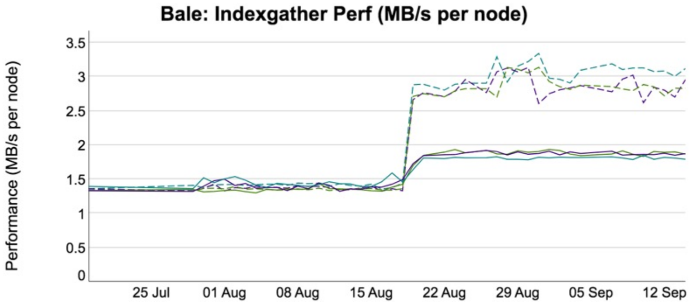


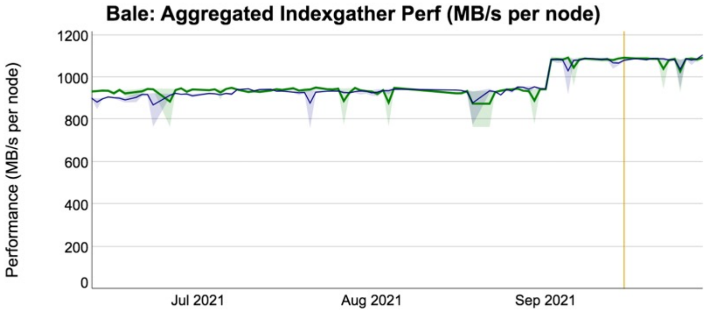












COMPILER IMPROVEMENTS

LLVM-BY-DEFAULT

Background:

- Traditionally, Chapel has generated C code as its “portable assembly”
  - LLVM-based back-end was also available as an option

In Chapel 1.25:

- Finally made good on a long-term intention to switch to our LLVM back-end by default
  - C-based compilation is still available as an option
- **Motivation:**
  - reduce burden of attempting to support all versions of all C compilers
  - communicate Chapel semantics more directly to back-end than C permits
  - leverage community investment in, and familiarity with, LLVM
  - somewhat faster compilation times, on average
  - attractive path for targeting GPUs

Status:

- A bit terrifying in the “what will users find in the field that we missed?” sense
  - but so far, no major fires

COMPILER REWORK: OFF TO A STRONG START

Background:

- The traditional Chapel compiler is…

  …slow (seconds to minutes)

  …often difficult to understand, in the presence of errors

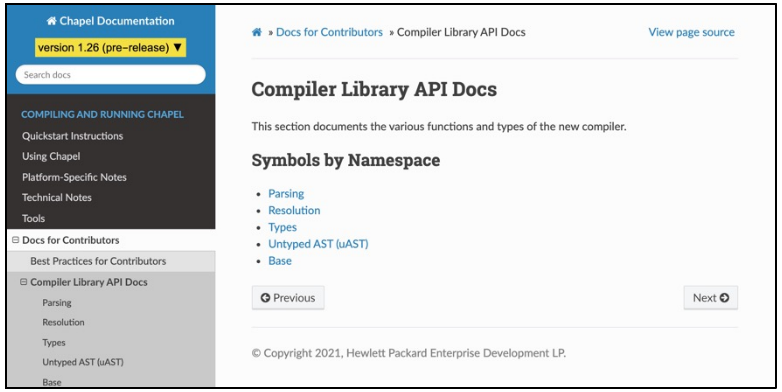
  …not terribly well-architected: inflexible, challenging for new contributors to get started
- These largely reflect its history as a scrappy research project, by a small team, moving fast

This Effort:

- This release, kicked off an effort to massively rework it and address these lacks:
  - faster / more flexible compilation: separate compilation, incremental recompilation, dynamic evaluation of code
  - better user experience
  - easier to contribute to

Status:

- parsing ~3/4 of Chapel features into the new internal representation (IR)
  - converting user code down to traditional compiler's IR and executing it
- first draft of resolution for types and calls
- code structure documented online: https://chapel-lang.org/docs/main/developer/compiler-internals/index.html



TARGETING GPUS WITH CHAPEL

GPUS: NOTIONAL GOAL

A Sample GPU Computation, notionally:

```
on here.GPU {
  var A = [1, 2, 3, 4, 5];
  forall a in A do
    a += 5;
}
```

GPUS: SIX MONTHS AGO

A Sample GPU Computation, as of Chapel 1.24:

```
pragma "codegen for GPU"
export proc add_nums(A: c_ptr(real(64))){
  A[0] = A[0]+5;
}

var funcPtr = createFunction();
var A = [1, 2, 3, 4, 5];
__primitive("gpu kernel launch", funcPtr,
            <grid and block size>,…,
            c_ptrTo(A), …);
writeln(A);
```

```
extern {
  #define FATBIN_FILE "chpl__gpu.fatbin"
  double createFunction(){
    fatbinBuffer = <read FATBIN_FILE into buffer>
    cuModuleLoadData(&cudaModule, fatbinBuffer);
    cuModuleGetFunction(&function, cudaModule,
                        "add_nums");}

}
```

Read fat binary and create a CUDA function

GPUS: TODAY

A Sample GPU Computation, in Chapel 1.25:

```
on here.getChild(1) {
  var A = [1, 2, 3, 4, 5];
  forall a in A do
    a += 5;
}
```

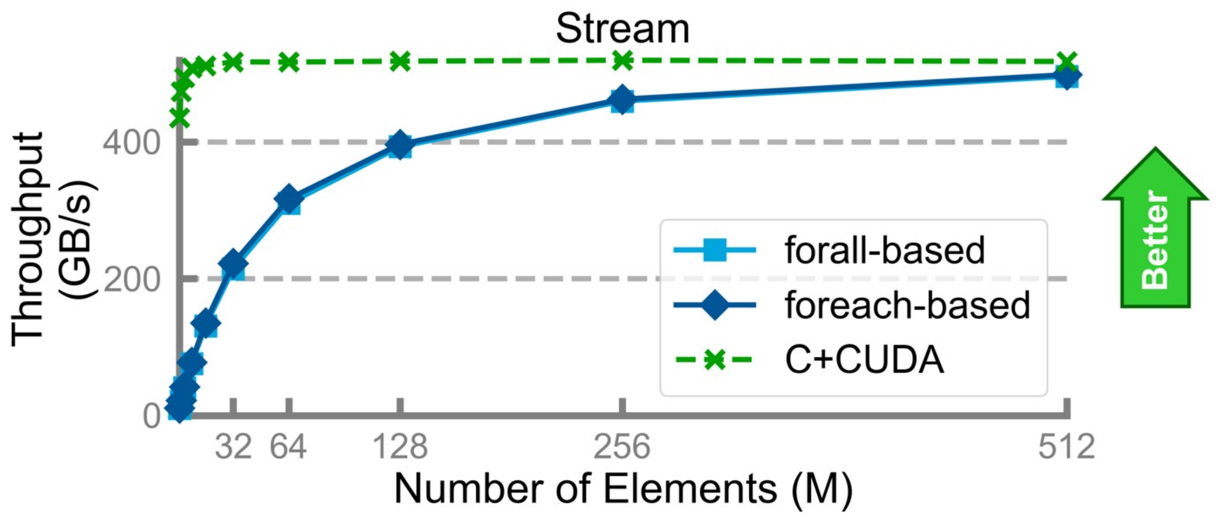
GPUS: INITIAL PERFORMANCE STUDY

HPCC Stream: very few changes needed to our typical Stream code to target GPUs

```
on here.getChild(1) {
  var A, B, C: [1..n] real;
  const alpha = 2.0;

  forall b in B do b = 1.0;
  forall c in C do c = 2.0;

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



GPUS: NEXT STEPS

- Plenty of housecleaning, refactoring, streamlining, etc.
- Language design issues
- Support richer and more flexible styles of programming
- Support a richer model of memory and inter-device data transfers
- Support a wider variety of vendors
- Further performance analysis and optimization

SUMMARY

SUMMARY

Great progress since Chapel 1.24:

- Performance and portability improvements
- Chapel 2.0 stabilization, especially w.r.t. libraries
- Massive strides in GPU support
- Strong start on compiler revamp
- User support and community activity, including CHIUW 2021

Near-term priorities:

- Continue with, and accelerate, library stabilization for Chapel 2.0
- Continue efforts to target GPUs and revamp the compiler
- Ongoing user support and outreach

THANK YOU

https://chapel-lang.org
@ChapelLanguage