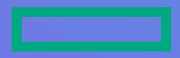


Hewlett Packard
Enterprise

Chapel's Performance Tracking System

Jade Abraham

July 10th, 2025

What is Chapel?

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative

chapel-lang.org

Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive

Overview

- Goals of performance tracking

- Configurations we test
  - Networks
  - Single node vs Multi node
  - Backend Compilers
  - Key User Applications

- Key Features
  - Graphs
  - Data storage
  - Custom benchmarking
  - Playgrounds

- Areas for improvement

Goals

- Track Benchmarks Written in Chapel
  - Regressions and improvements
  - Need ability to triage and learn from regressions
  - Also need to understand what causes improvements, can that drive future improvements?

- Track Key User Applications
  - We support users, we should find performance regressions before they do
  - Arkouda, CHAMPS, ChOp, miniBUDE

- Track Compiler Performance
  - Prevent compilation time/memory consumption from regressing

- Transparency
  - All our perf results are publicly available, the good and the bad
  - https://chapel-lang.org/perf-nightly.html

Current Configurations (Non-exhaustive list)

- Single Node (24 cores and 256 GB RAM)
  - Default Configuration
  - C Backend – GNU and clang
  - Compiler Performance – Default and Optimized
  - Arkouda (User Application)
- HPE Slingshot runs (200 Gb/s Slingshot 11, 64 cores and 512 GB RAM)
  - 16 Node
  - 1 Node
- InfiniBand runs (200 Gb/s HDR, 72 cores and 512 GB RAM)
  - 16 Node
  - 16 Node - Arkouda (User Application)
  - 16 Node - CHAMPS (User Application)

Current Configurations (Non-exhaustive list)

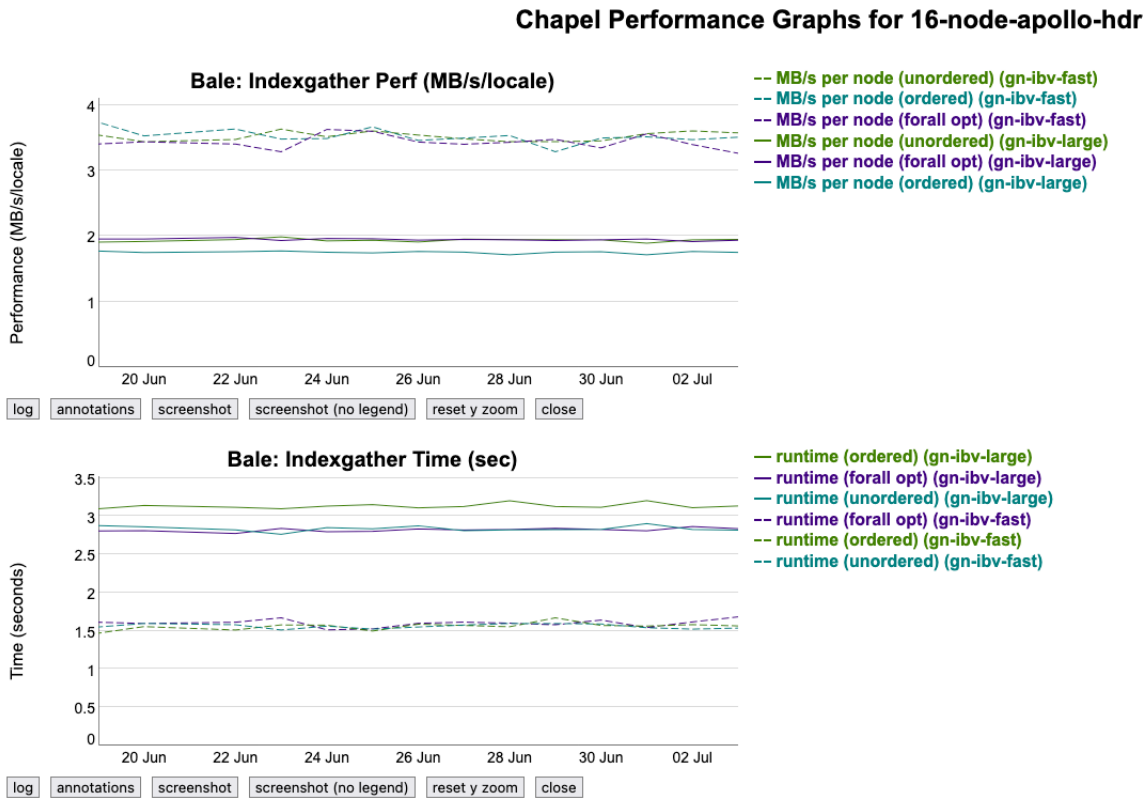
- Cray XC (node's have 48 cores and 192 GB RAM)
  - 16 Node
  - 1 Node
  - 16 Node - Arkouda (User Application)
- Single Node GPU
  - Nvidia A100 (4 GPUs, 64 CPUs)
  - AMD MI250x (4 GPUs, 64 CPUs)

Graphs

- Graphs are described in .graph files
- A single graph file can pull in multiple data files

- Generation process
  - Python script to parse all graph files + data + annotations into graphdata.js
  - Custom web scripting to generate graphs (dygraph)

- Annotations
  - YAML file of annotations for graphs
  - Attach annotations to entire configurations or individual tests



https://chapel-lang.org/perf/16-node-apollo-hdr/?graphs=baleindexgatherperfmbslocale,baleaggregatedindexgatherperfmbslocale,baleindexgathercompileroptvariantsperfmbspernode,baleindexgatherexplicitoptvariantsperfmbspernode,baleindexgathertimesec

Live Graphs!

- Single Node: https://chapel-lang.org/perf/chapcs/
  - Shootout
  - Perf Tracking – Last Two Weeks
- 16 Node InfiniBand: https://chapel-lang.org/perf/16-node-apollo-hdr/
  - Selected graphs

Infrastructure

- Chapel's test system supports correctness and performance tests
- Each nightly job is configured by
  - Public scripts in the Chapel repo
  - Internal Jenkins configuration
- Jenkins jobs run nightly on hardware hosted by HPE

util > cron > $ test-perf.chapcs.bash

```
1  #!/usr/bin/env bash
2  #
3  # Run performance tests on a chapcs machine with default configuration.
4
5  UTIL_CRON_DIR=$(cd $(dirname ${BASH_SOURCE[0]}) ; pwd)
6
7  export CHPL_TEST_PERF_CONFIG_NAME='chapcs'
8
9  source $UTIL_CRON_DIR/common-perf.bash
10
11 export CHPL_LAUNCHER=none
12
13 export CHPL_NIGHTLY_TEST_CONFIG_NAME="perf.chapcs"
14
15 $UTIL_CRON_DIR/nightly -cron -performance -releasePerformance -numtrials 5 -startdate
   09/10/15
16
```

- Data
  - Data is saved into tab separated .dat files
  - Sync'd to a GitHub repository for historical records

- Graphs
  - Graphs are regenerated each night by the Jenkins job
  - Separate Jenkins job collects graphs and automatically uploads them to webhost

Writing Performance Tests

- Most performance tests do not require any custom scripting
- Each test is a Chapel file with some extra config files
  - perfcompopts
    - Compilation options passed to the Chapel compiler
  - perfexecopts
    - Options passed to the compiled executable
  - perfkeys
    - Defines characteristics to extract from test for data files
    - Support for verifying correct output from test in same perf test
- Built in support for testing other languages for comparison, mostly C
  - Can test just about anything via custom scripting

- We can also plug in arbitrary scripts to test larger/more complex projects
  - This is the strategy for Arkouda and CHAMPS

Example Test: test/npb/ep/mcahir/ep.chpl

- ep.perfexecopts: defines multiple variations on the same benchmark

test > npb > ep > mcahir > ep.perfexecopts

```
1  --probClass=S # ep.S
2  --probClass=W # ep.W
3  --probClass=A # ep.A
4  --probClass=B # ep.B
5  #--probClass=C # ep.C
6  #--probClass=D # ep.D
```

- ep.perfkeys: for each variation run, defines the performance keys to look for

test > npb > ep > mcahir > ep.perfkeys

```
1  Time in seconds =
2  Mop/s total     =
3  Mop/s/process   =
4  Verification    =
5  verify:-1: ^ Verification .*SUCCESSFUL$
6
```

Example Test: test/npb/ep/mcahir/ep.chpl

- ep.S.dat: one of the data files generated

ep.S.dat

	# Date	Time in seconds =	Mop/s total =	Mop/s/process =	Verification =
2	09/10/15	0.08683	386.438	386.438	SUCCESSFUL
3	09/10/15	0.092351	363.336	363.336	SUCCESSFUL
4	09/10/15	0.090567	370.493	370.493	SUCCESSFUL
5	09/10/15	0.095063	352.97	352.97	SUCCESSFUL
6	09/10/15	0.094059	356.738	356.738	SUCCESSFUL
7	09/11/15	0.088565	378.868	378.868	SUCCESSFUL
8	09/11/15	0.09379	357.761	357.761	SUCCESSFUL
9	09/11/15	0.089163	376.327	376.327	SUCCESSFUL
10	09/11/15	0.088058	381.049	381.049	SUCCESSFUL
11	09/11/15	0.088211	380.388	380.388	SUCCESSFUL
12	09/12/15	0.083856	400.143	400.143	SUCCESSFUL

Example Test: test/npb/ep/mcahir/ep.chpl

- ep.graph: defines how the data is presented

test > npb > ep > ep.graph

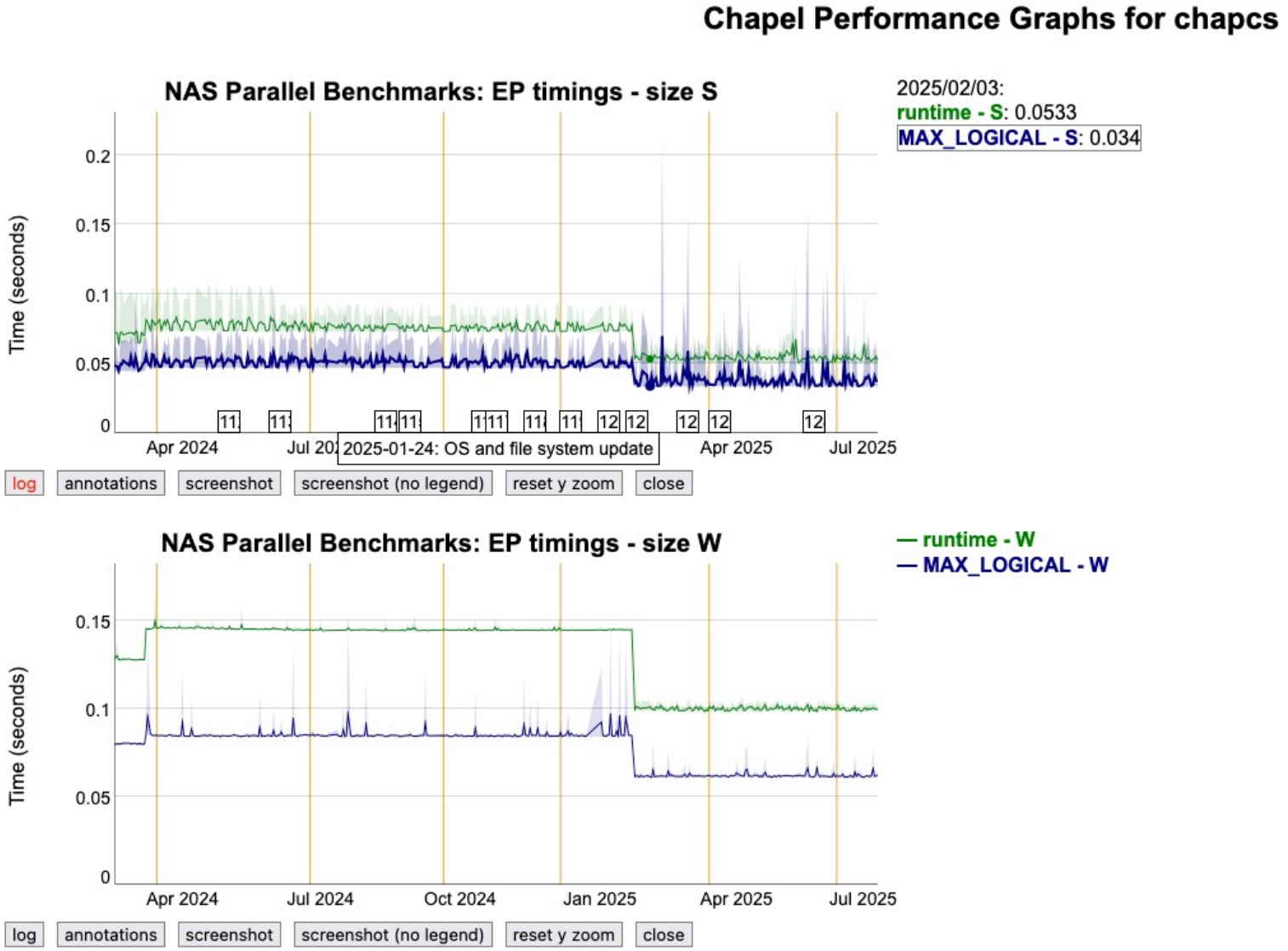
```
perfkeys:  Time in seconds =, Time in seconds =
files: ep.S.dat, ep.S.maxLogical.dat
graphkeys: runtime - S, MAX_LOGICAL - S
graphtitle: NAS Parallel Benchmarks: EP timings - size S
ylabel: Time (seconds)

perfkeys:  Time in seconds =, Time in seconds =
files: ep.W.dat, ep.W.maxLogical.dat
graphkeys: runtime - W, MAX_LOGICAL - W
graphtitle: NAS Parallel Benchmarks: EP timings - size W
ylabel: Time (seconds)

perfkeys:  Time in seconds =, Time in seconds =
files: ep.A.dat, ep.A.maxLogical.dat
graphkeys: runtime - A, MAX_LOGICAL - A
graphtitle: NAS Parallel Benchmarks: EP timings - size A
ylabel: Time (seconds)

```

Example Test: test/npb/ep/mcahir/ep.chpl



https://chapel-lang.org/perf/chapcs/?graphs=nasparallelbenchmarkseptimingssizes,nasparallelbenchmarkseptimingssizew,nasparallelbenchmarkseptimingssizea,nasparallelbenchmarkseptimingssizeb

Playgrounds

- We have two playgrounds
  - Single node
  - 16 node InfiniBand

- Allows developers to test new features or dependencies without disrupting main

- Example: https://chapel-lang.org/perf/chapcs/llvm-16/?graphs=hpccratime

11 util/cron/test-perf.chapcs.playground.bash

```
20  20
21  21  export CHPL_NIGHTLY_TEST_CONFIG_NAME="perf.chapcs.playground"
22  22
23  23  #
24  24  # 0) Update the comment that follows this block to describe what's being tested
25  25  # 1) Update GITHUB_USER to the GitHub username owning the branch
26  26  # 2) Update GITHUB_BRANCH to the name of the GitHub branch under that user
27  27  # 3) Update SHORT_NAME to be a short name that will be used in graph generation
28  28  # 4) Update START_DATE to be today, using the format mm/dd/yy
29  29  #
30  30
31      - GITHUB_USER=chapel-lang
32      - GITHUB_BRANCH=main
33      - SHORT_NAME=libmvec
34      - START_DATE=04/16/25
    31  + GITHUB_USER=jabraham17
    32  + GITHUB_BRANCH=llvm-20
    33  + SHORT_NAME=llvm20
    34  + START_DATE=04/30/25
```

Triage

- Weekly
  - Several members of the Chapel team meet to review the last week's performance data
  - Look at key configs and user applications for regressions/improvements
  - Assign people to reproduce and diagnose changes

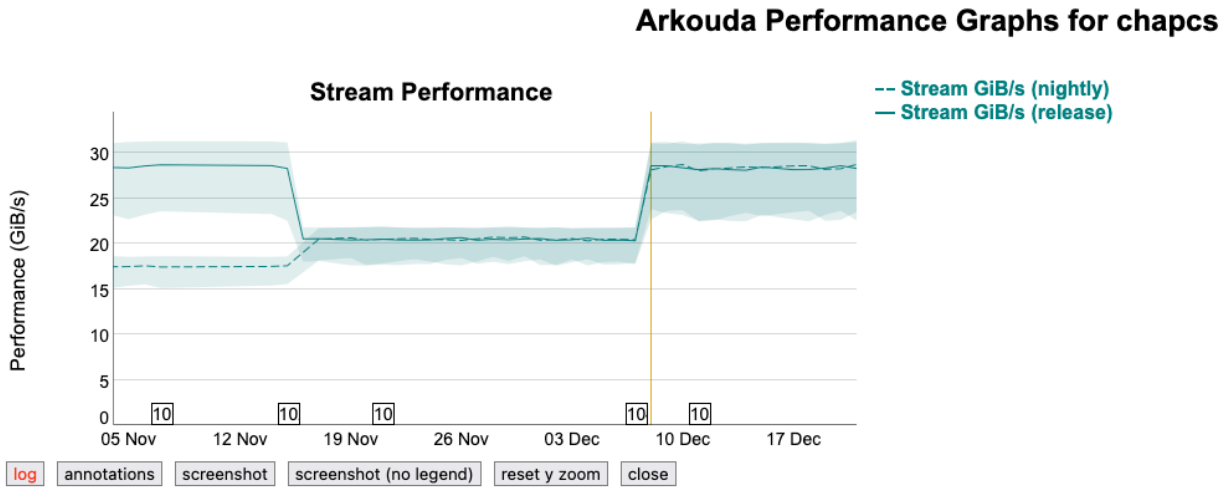
- Before every release
  - The Chapel team goes through the performance data since the previous release (~3 months)
  - Look at all configs and check for anything that was missed

Recent Case Study #1

- Arkouda developer added new support for multi-dimensional arrays

- Array creation regressed
  - We could immediately tell the problem was user created by testing both Chapel nightly and release
  - https://chapel-lang.org/perf/arkouda/chapcs/?startdate=2023/11/04&enddate=2023/12/21&graphs=streamperformance

- Developer was able to diagnose the root cause and fix the regression

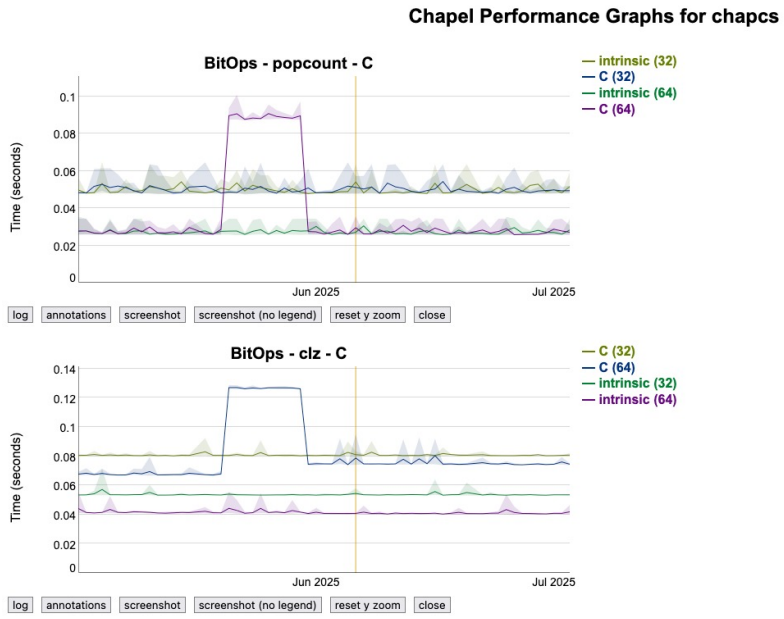


Recent Case Study #2

- We upgraded our nightly system to use LLVM 20

- Bitops comparison test regressed
  - The test (written in C) is a microbenchmark of bitwise operations in the Chapel runtime
  - https://chapel-lang.org/perf/chapcs/?startdate=2025/05/02&enddate=2025/07/03&graphs=bitopspopcountc,bitopsclzc,bitopsctzc

- Found a LLVM bug and a bug in the performance test
  - https://github.com/llvm/llvm-project/issues/142042
  - https://github.com/chapel-lang/chapel/pull/27308



Areas of Improvements

- New Features
  - Add git SHA next to the date in the graphs
  - Add information on what file and command line options create a given line on a graph
  - Split data files on more than just a per-file basis
    - Adding/removing a perf key to an existing file today requires editing the historical data
  - Ability to setup perf tests from a single configuration file, rather than multiple files
  - Arrows to show where a regression is
    - Some graphs are “up is better”, some are “down is better”. This routinely trips people up.

- Modernizing our testing infrastructure
  - We have infrastructure that works for us and provides results, but it is built on aging software
  - Many scripts are 20+ years old and few people know how they work
  - We rely heavily on bespoke solutions rather than standard tools
  - Managing Jenkins and system configurations is currently a big time sink

Thank you

https://chapel-lang.org
@ChapelLanguage