Benchmarks and Performance Optimizations

Chapel version 1.20

September 19, 2019

chapel_info@cray.com

chapel-lang.org

@ChapelLanguage

CRAY®

a Hewlett Packard Enterprise company

CRAY CHAPEL

Outline

- Unordered Copy Improvements
- Unordered Compiler Optimization
- Task Placement Improvements
- InfiniBand Improvements
- Performance Portability
- RandomStream Optimization
- Memory Pressure
- Memory Leaks

Unordered Copy Improvements

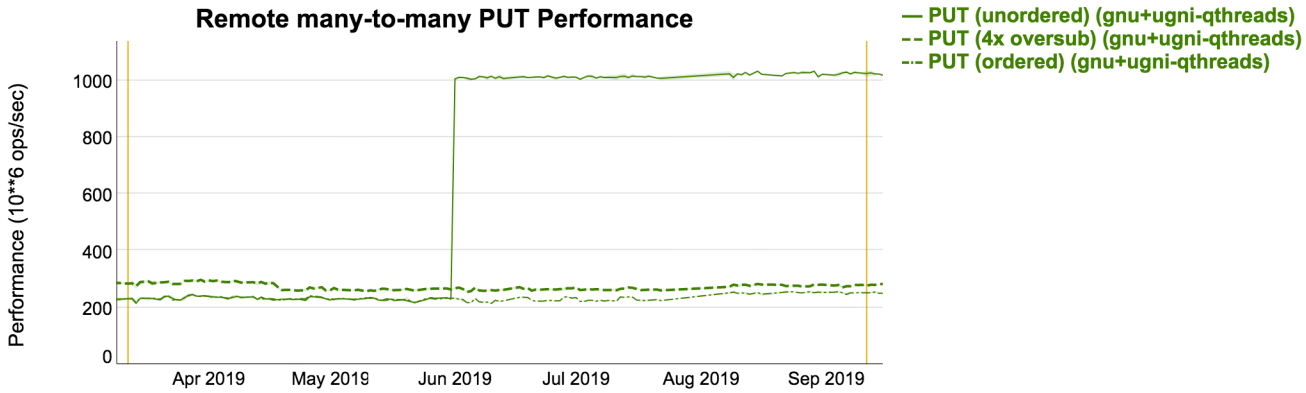
Unordered Copy: Background

- 'unorderedCopy(dst, src)' was added in 1.19
  - Provides faster remote copies when sequential-consistency is not required

- Initial implementation had some limitations
  - Only optimized when source was remote (network GET)
  - Source had to be a 'var' ('const' / 'param' not supported)
  - Did not work for promoted calls
  - Only supported numeric types

Unordered Copy: This Effort

- Optimized performance when destination is remote (network PUT)
  - 4.5x faster than ordered PUTs



Unordered Copy: This Effort

- Extended support to 'param' and 'const' sources:

```
forall i in 1..n do
  unorderedCopy(A[i], 1);   // 1 is a ‘param’ value
forall i in 1..n do
  unorderedCopy(A[i], i);   // i is a ‘const’ variable
```

- Fixed support for promoted calls:

```
unorderedCopy(A, 1);
unorderedCopy(A, B);
```

- Extended support to Boolean types:

```
unorderedCopy(done, true);
```


Unordered Copy: Impact, Next Steps

Impact:

- unorderedCopy is now more usable and performant

Next Steps:

- Extend unorderedCopy to all plain old data (POD) types
  - Tune performance for larger POD types
- Optimize for all comm layers
  - Currently only optimized for comm=ugni

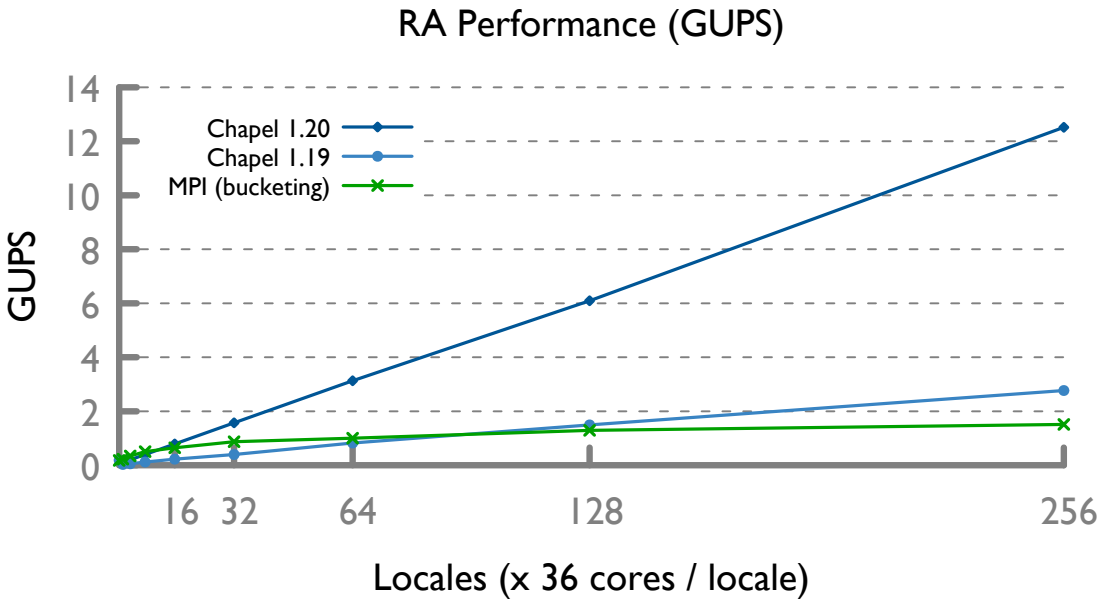
Unordered Compiler Optimization

Unordered Optimization: Background

- Unordered operations provide a significant performance speedup
  - But they are an advanced feature that break the memory consistency model
- Many Chapel idioms should be amenable to unordered operations
  - Chapel semantics make these types of transformations possible
    - e.g., most synchronization-free forall loops
- 1.19 included a prototype optimization to automatically use unordered ops
  - Implemented towards end of release cycle, so left "off" by default

Unordered Optimization: This Effort

- Improved the optimization to support more distributions
- Enabled the unordered compiler optimization by default



Unordered Optimization: Impact, Next Steps

CRAY
a Hewlett Packard Enterprise company

Impact:

- Many programs benefit from unordered operations with no source changes

Next Steps:

- Extend optimization to handle promoted expressions
- Expand how many statements can be optimized
  - Currently, only the last statement of a forall loop is optimized

Task Placement Improvements

Task Placement: Background

- Chapel relies on first touch to get correct NUMA affinity
  - First store to a memory page gives it affinity with the CPU's NUMA domain
- Computations with tasking patterns that match initialization get proper affinity
  - i.e., tasks will operate on memory local to their NUMA domain

```
var A, B, C: [1..m] real;                  // Parallel initialization

forall (a, b, c) in zip(A, B, C) do        // Same affinity as initialization, good performance
  a = b + alpha * c;
```


Task Placement: Background

- Improper affinity severely hurts memory bandwidth-driven benchmarks
  - Previously a simple round-robin assignment of tasks to threads was used
    - Only worked well if loops used all threads with no intervening tasks

```
var A, B, C: [1..m] real;                  // Parallel initialization

forall (a, b, c) in zip(A, B, C) do        // Same affinity as initialization, good performance
  a = b + alpha * c;

coforall 1..here.maxTaskPar/2 { }          // Interleaved tasks skew round-robin assignment

forall (a, b, c) in zip(A, B, C) do        // Tasks offset from initialization affinity, poor performance
  a = b + alpha * c;
```

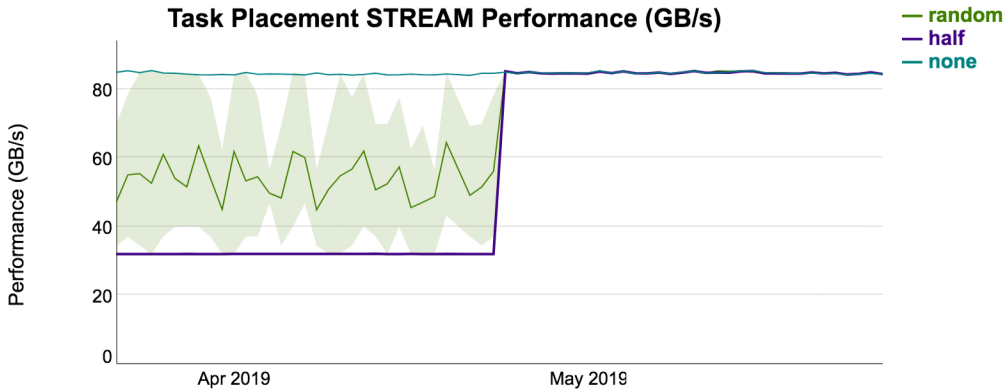

Task Placement: This Effort, Impact

This Effort: Implemented a task resetting policy

- When a parallel loop uses all threads, reset task placement

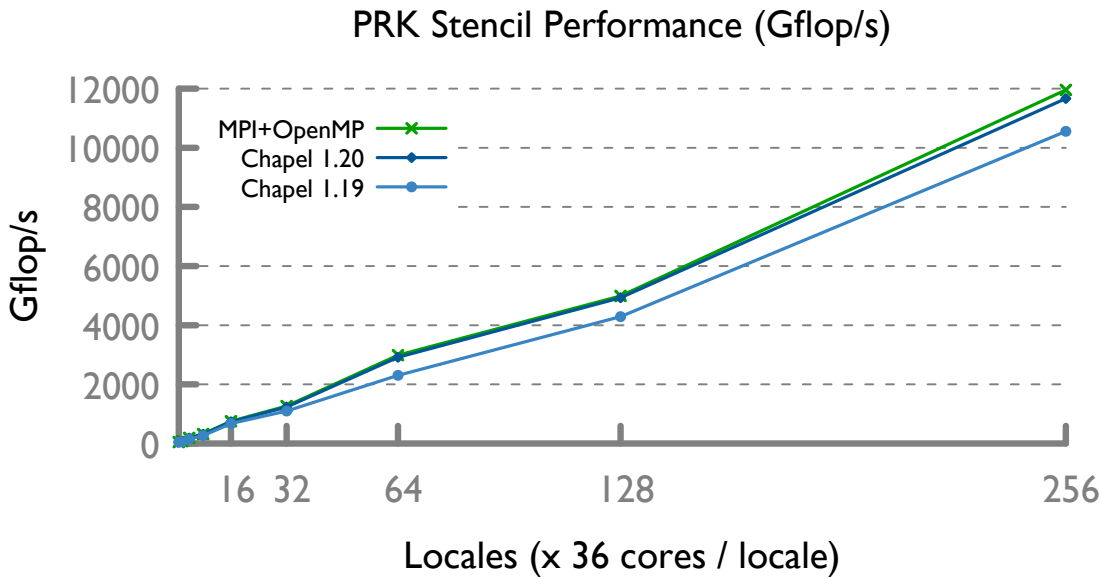
Impact: Improved performance for cases where interleaving tasks are created

- Up to a 2x improvement for Stream variation that created additional tasks



Task Placement: Performance Impact

- Improved performance for cases where interleaving tasks are created
  - 10% improvement for PRK Stencil



Task Placement: Next Steps

- Consider running with a process per NUMA domain
  - First touch is not always enough

```
var A, B, C: [1..m] real;          // parallel first touch

for (a, b, c) in zip(A, B, C) do   // serial computation, 1/2 ops on wrong NUMA domain
  a = b + alpha * c;

forall i in 1..m by -1 do.         // different parallel iteration, all ops on wrong NUMA domain
  A[i] = B[i] + alpha * C[i];
```

- Tradeoffs between performance and ease-of-use
  - Current locale-per-node mapping is easy to reason about
    - Intuitive transition from single- to multi-locale programs

InfiniBand Improvements

InfiniBand: Background

- To date, we have mostly focused on performance for Cray systems with ugni
  - Intent was to ensure we have the right language features/semantics first
    - Then expand capabilities to other networks

- Recently there have been several requests for better InfiniBand performance

InfiniBand: This Effort

- Added nightly InfiniBand performance testing
  - Results are available at: https://chapel-lang.org/perf/16-node-cs/
  - Tracking GASNet InfiniBand for segment large (default) and segment fast
    - Segment large does dynamic registration, fast does static
    - NUMA affinity is better with segment large
    - Communication performance is better with segment fast

config	Stream	PRK-Stencil	ISx	RA-rmo	RA-on
gn-ibv-large	~1500 GB/s	~780 GFlops/s	~55 s	0.0011 GUPS	0.0010 GUPS
gn-ibv-fast	~650 GB/s	~340 GFLops/s	~16 s	0.0021 GUPS	0.0010 GUPS

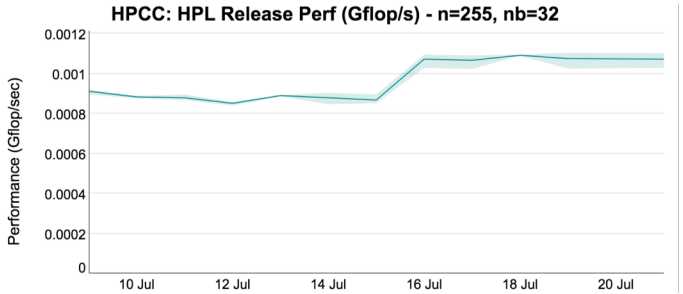
InfiniBand: This Effort, Impact

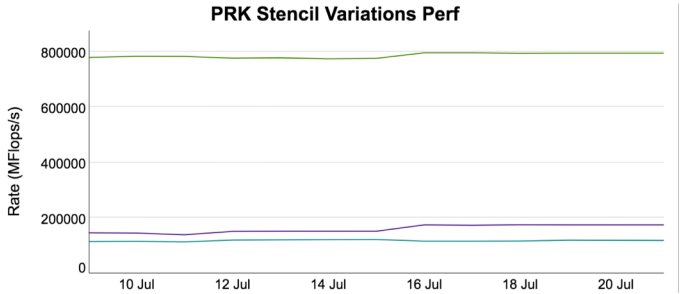
This Effort: Identified interference from the progress thread (on-stmt handler)

- Switched from an active to a blocking progress thread

Impact: Improved performance for compute-bound benchmarks

- Slightly increased latency for processing incoming on-stmts





InfiniBand: Next Steps

- Collaborate with the GASNet team and improve InfiniBand performance
  - Map to GASNet-EX remote atomics
  - Use multiple endpoints as they come online (better comm concurrency)
  - Explore better dynamic registration options to improve NUMA affinity

- Run reference benchmarks to figure out what InfiniBand is capable of
  - Use benchmarks where Chapel lags to guide further optimization

Performance Portability

Performance Portability: Background, Effort

Background: We want Chapel to perform well on all systems

- Performance on Cray networks is best and we have started to tune InfiniBand
- Little effort has been put into tuning Ethernet so far

This Effort: Ran core benchmarks on several networks to compare performance

- Cray Aries (ugni)
- FDR InfiniBand (gasnet-ibv segment large)
- Gigabit Ethernet (gasnet-udp segment everything)

Performance Portability: Status

- Core benchmarks on a Cray-XC and Cray-CS
  - Similar per-node hardware: dual 18-core Broadwell CPUs and 128 GB RAM

config	Stream	PRK-Stencil	ISx	RA-rmo	RA-on
ugni (Aries)	~1500 GB/s	~780 GFlops/s	~9 s	0.1150 GUPS	0.0100 GUPS
gn-ibv-large (56 Gb FDR)	~1500 GB/s	~780 GFlops/s	~55 s	0.0011 GUPS	0.0010 GUPS
gn-udp (1 Gb E)	~1500 GB/s	~450 GFlops/s	~800 s	0.0003 GUPS	0.0004 GUPS

Performance Portability: Next Steps

- Add nightly Ethernet performance testing and tune performance
- Run reference benchmarks for Ethernet and InfiniBand
- Explore optimizations that will benefit high-latency networks
  - Cray networks have extremely low latency, especially for small transfers
  - Other networks will likely require some sort of aggregation

RandomStream Optimization

RandomStream: Background, This Effort

Background:

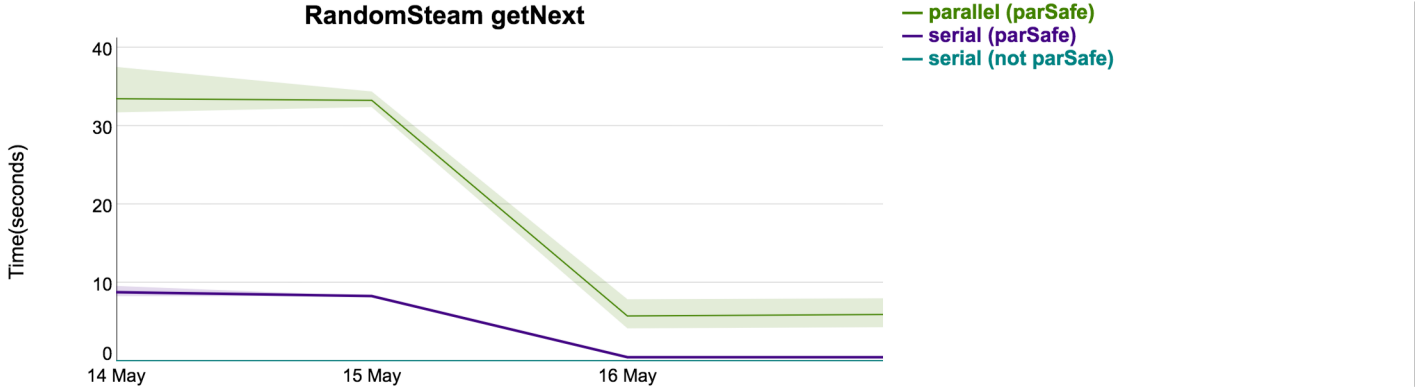
- RandomStream() library routine is parallel-safe by default
- Previously, a sync was used as a lock to provide parallel safety
  - Sync variables are heavyweight
  - ~200x overhead when parallel safety is not required

This Effort:

- Created an optimized test-and-test-and-set spinlock
- Replaced RandomStream sync lock with optimized spinlock
- Used spinlock to optimize other parallel-safe data structures

RandomStream: Impact

- Significantly faster RandomStream()
  - Only 25% overhead for parallel safety when generating real(64) values



RandomStream: Next Steps

- Reduce overhead for sync variables
  - Minimize cost for uncontended accesses
- Create a lock/mutex library
  - Include spinlock and more sophisticated/scalable locks

Memory Pressure Improvements

Memory Pressure: Background

- By default, Chapel task stacks are 8MB
  - Chosen as a "safe" default to help avoid stack overflows
- Normally, only pages that are used are backed by physical memory
- On networks that require registration, full 8MB is backed by physical memory
  - This limits the number of tasks that can exist/run concurrently

Memory Pressure: Background

- Creating many tasks with no user-induced yield could also exhaust memory
  - Tasks should run to completion before another is started
  - However, the following no-op program used to run out of memory under ugni

```
coforall 1..100_000 do on Locales[1] { }
```

- This would happen because:
  - Locale 0 enqueues tasks on Locale 1
  - Tasks created on Locale 1 yield during implicit comm that signals completion
  - Yielded tasks are moved to the end of a task queue
  - Task scheduler round-robins through all tasks, whether new or yielded

Memory Pressure: Effort, Next Steps

This Effort:

- Avoid yielding until a task has performed a non-trivial amount of comm
  - Reduces memory pressure from short tasks

Next Steps:

- Consider more sophisticated task queueing strategies
- Explore ways to reduce stack size to increase number of concurrent tasks
  - e.g., compiler analysis to determine maximum stack size

Memory Leak Improvements

Memory Leaks: Background, This Effort

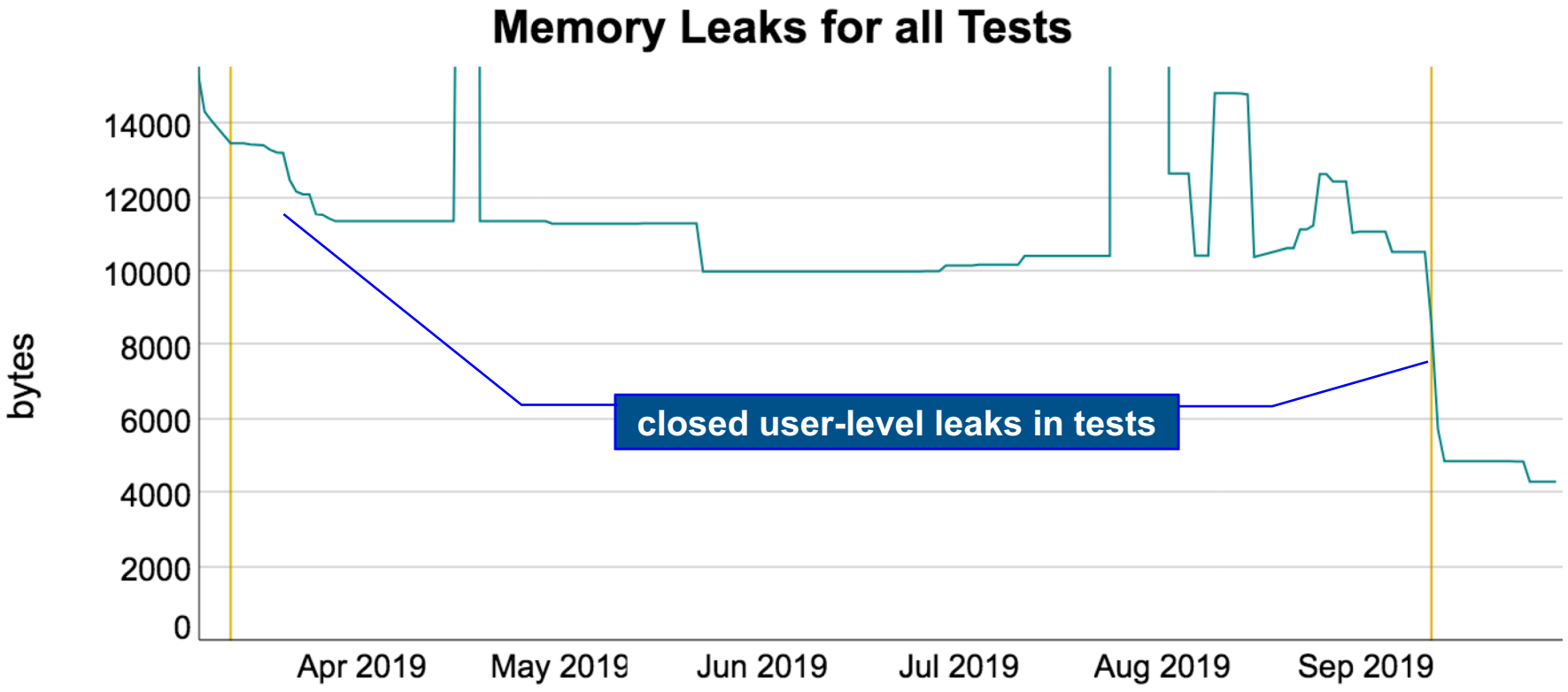
Background:

- Historically, Chapel testing resulted in a large number of memory leaks
- Recent releases have driven this number sharply downward

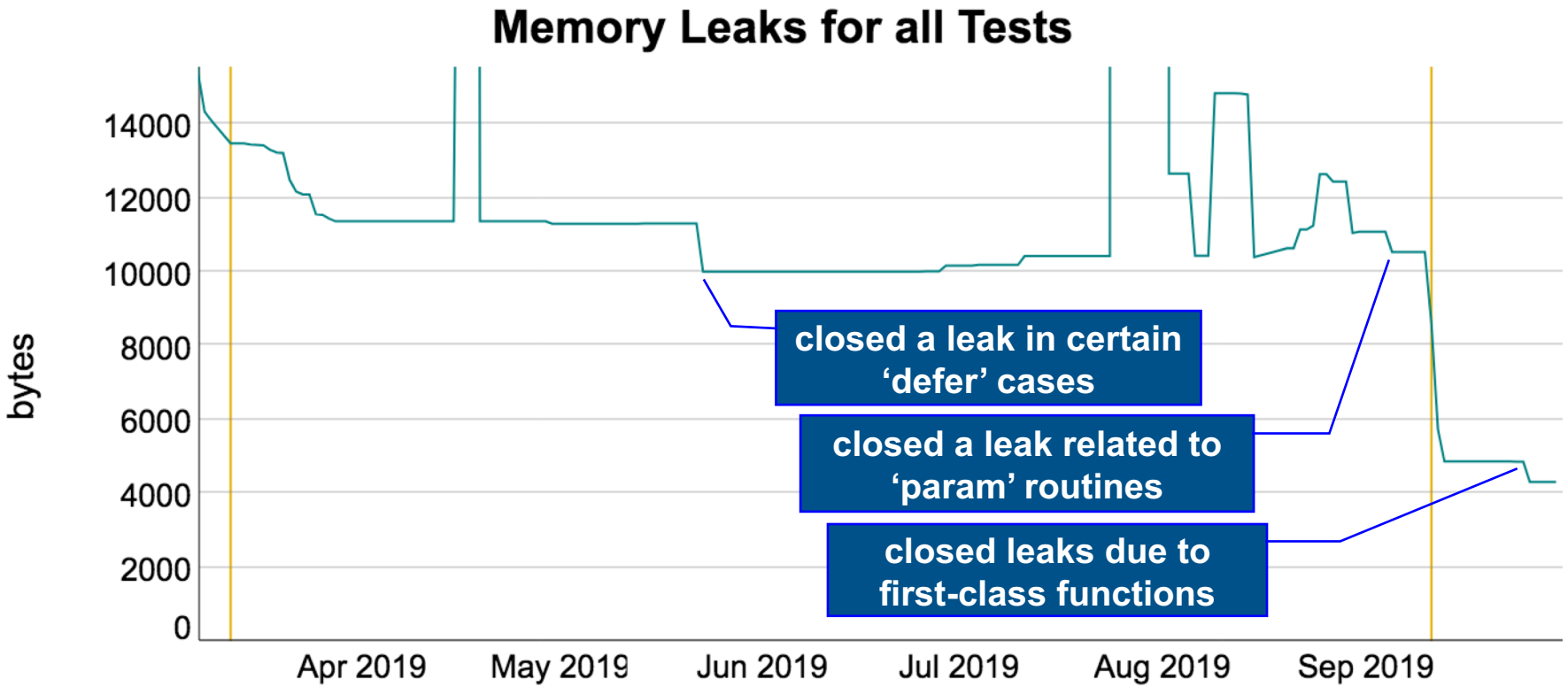
This Effort:

- Memory leaks were not a significant priority for this release
- However, additional progress was made, most notably:
  - Closing leaks in tests themselves
  - Closing leaks caused by defer statements
  - Just after the release, closed leaks due to first-class functions

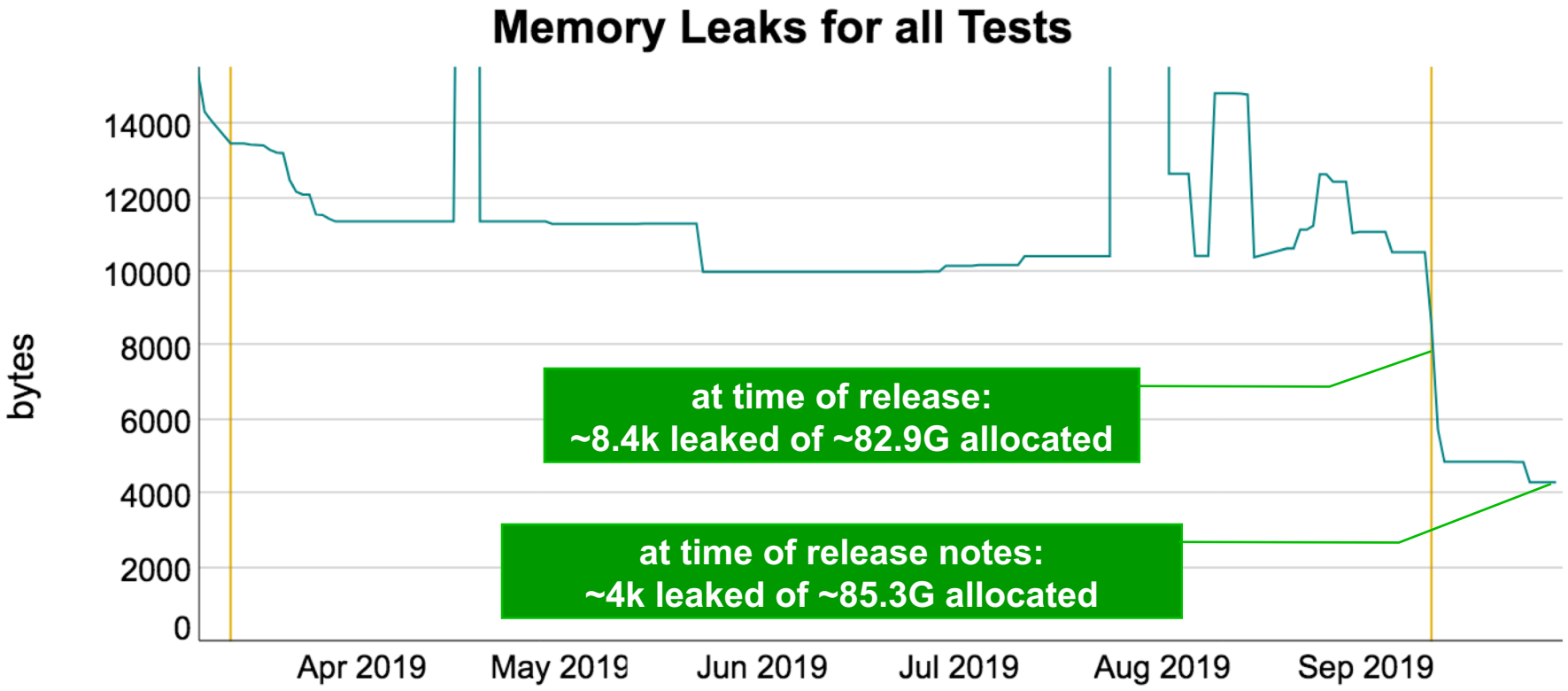
Memory Leaks: This Effort, Impact (major fixes)



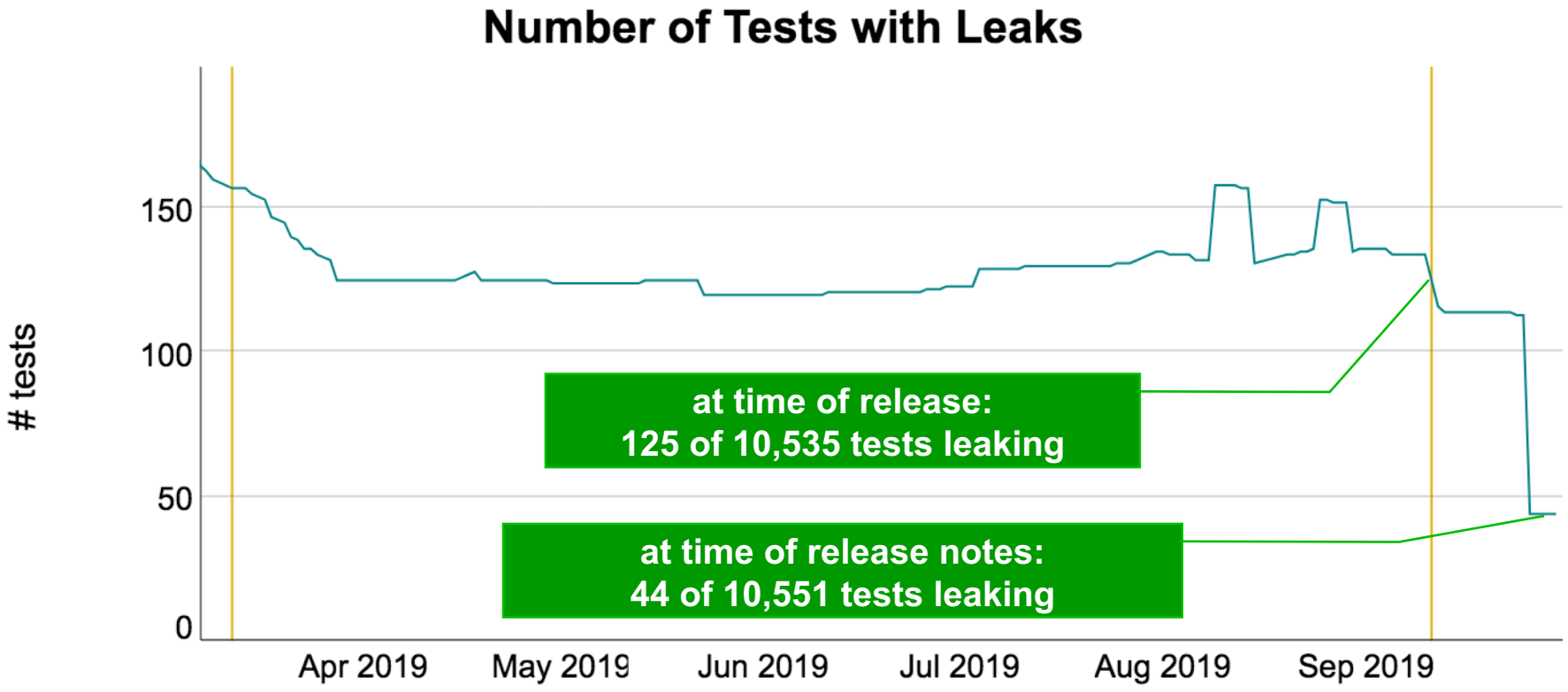
Memory Leaks: This Effort, Impact (major fixes)



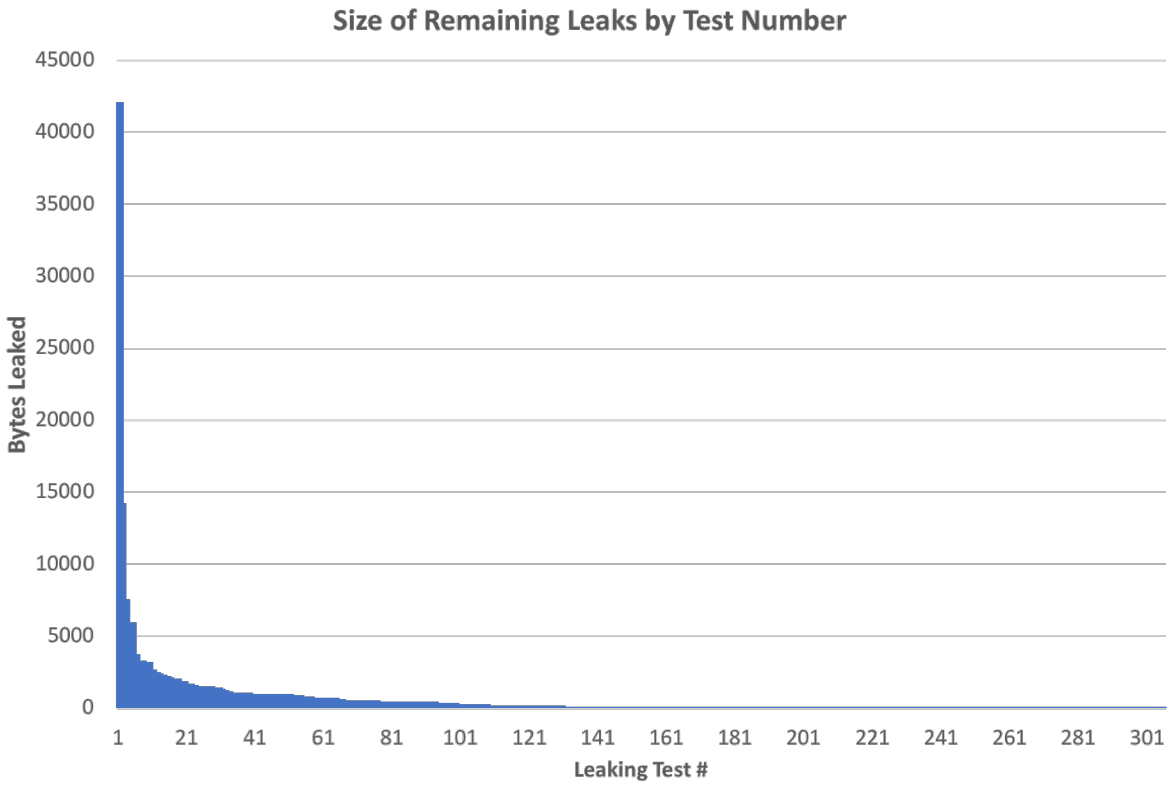
Memory Leaks: Status (amount of memory)



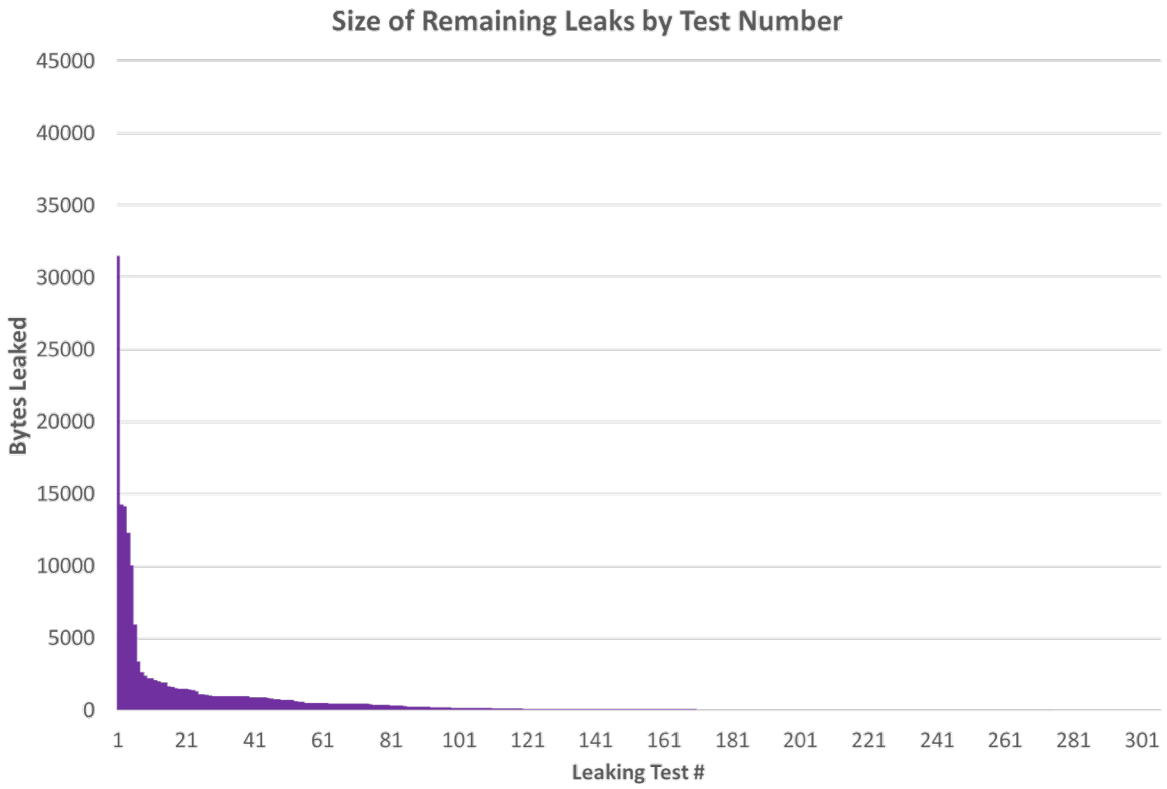
Memory Leaks: Status (number of tests)



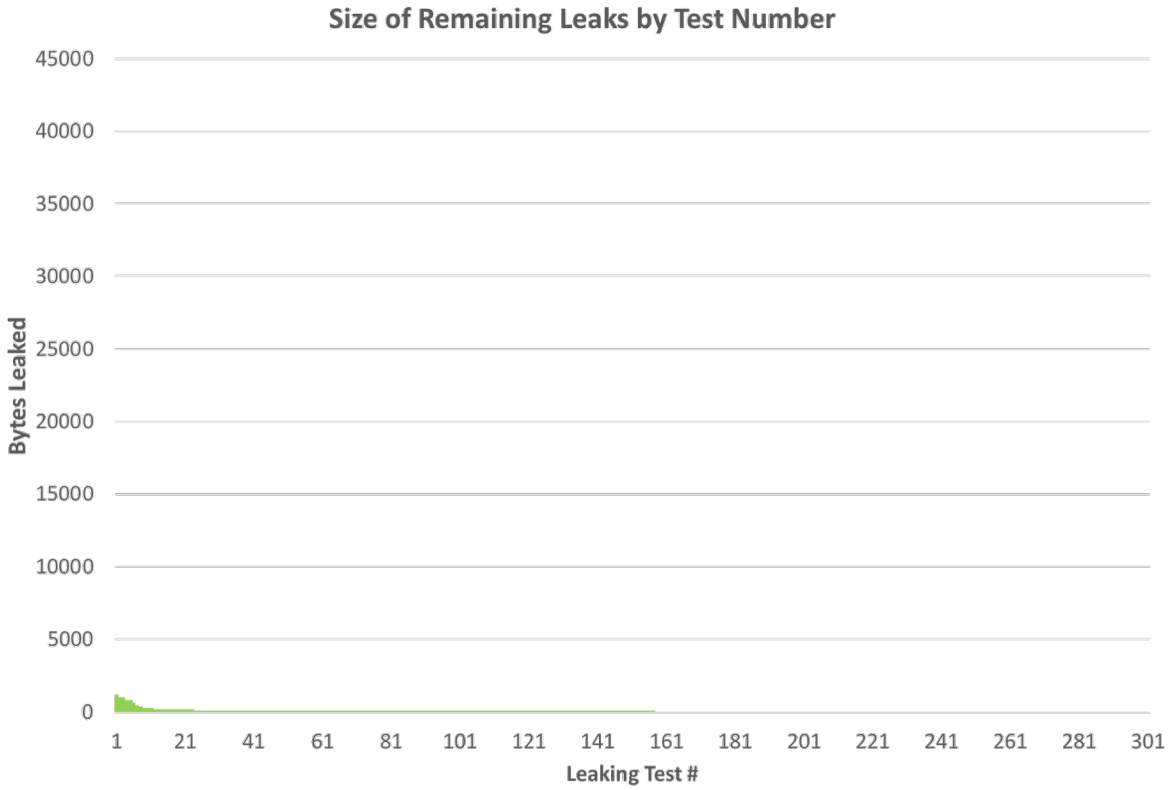
Memory Leaks: Remaining Leaks (as of 1.17)



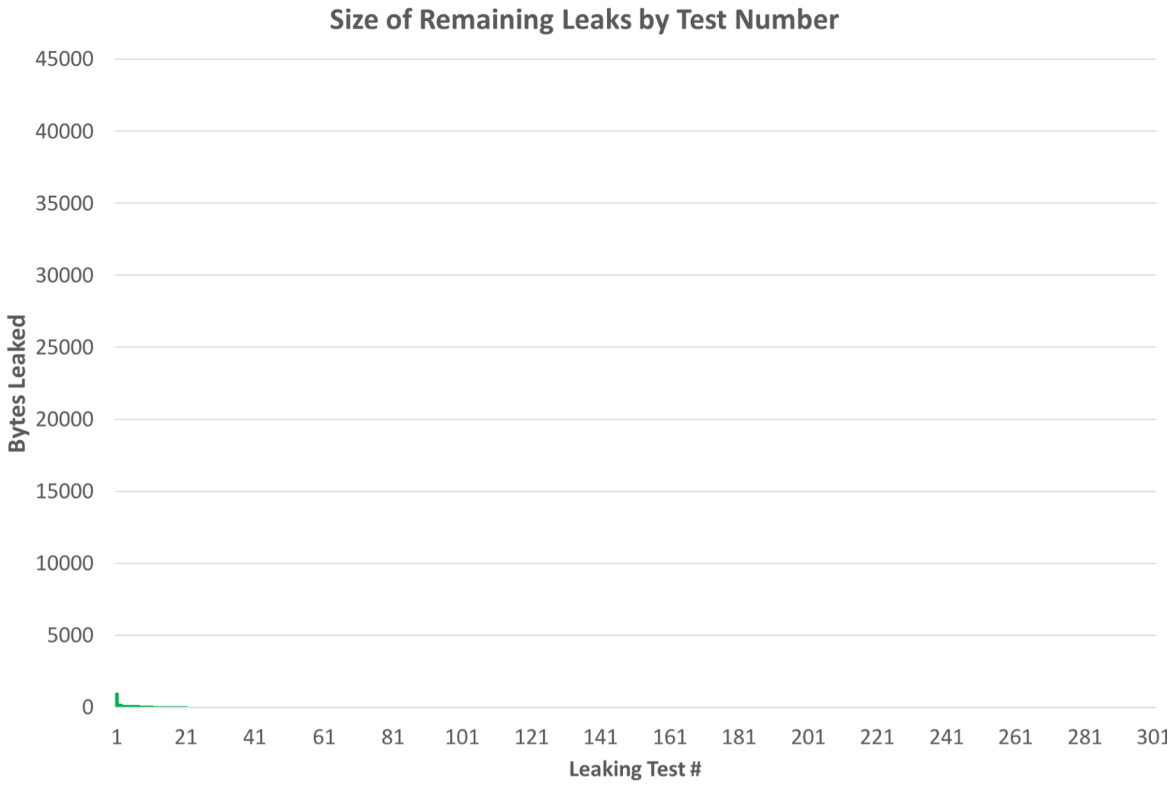
Memory Leaks: Remaining Leaks (as of 1.18)



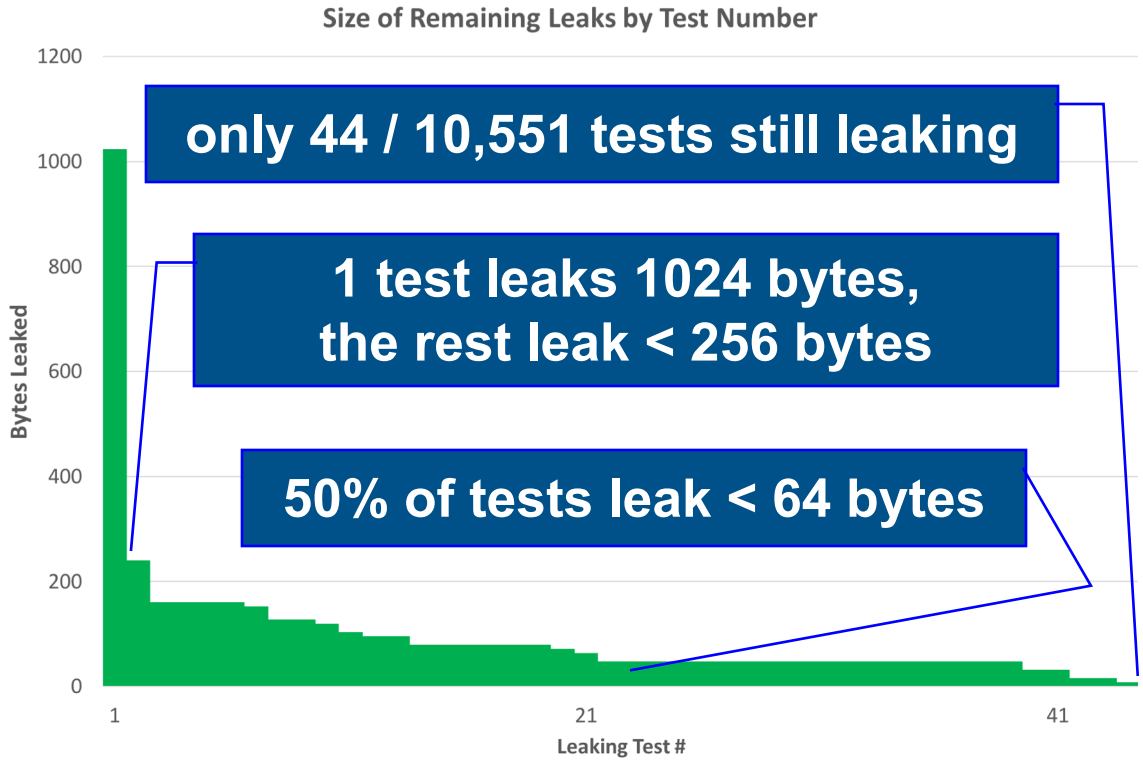
Memory Leaks: Remaining Leaks (as of 1.19)



Memory Leaks: Remaining Leaks (as of 1.20)



Memory Leaks: Remaining Leaks (as of 1.20)



Memory Leaks: Next Steps

- Close remaining leaks:
  - A few cases of leaks in error-handling
  - A few cases of leaking for certain domain map patterns
  - A few other corner cases
- Turn nightly leaks testing into a correctness, rather than performance, check
- Check status of leaks for multi-locale runs, LLVM back-end

For More Information

For a more complete list of related changes in the 1.20 release, refer to the 'Performance Improvements' and 'Memory Improvements' sections in the CHANGES.md file

FORWARD LOOKING STATEMENTS

This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries ("Hewlett Packard Enterprise") may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties' control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray's business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise's businesses; the competitive pressures faced by Hewlett Packard Enterprise's businesses; risks associated with executing Hewlett Packard Enterprise's strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise's other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.

THANK YOU

QUESTIONS?


chapel_info@cray.com


@ChapelLanguage


chapel-lang.org

cray.com


@cray_inc


linkedin.com/company/cray-inc-/