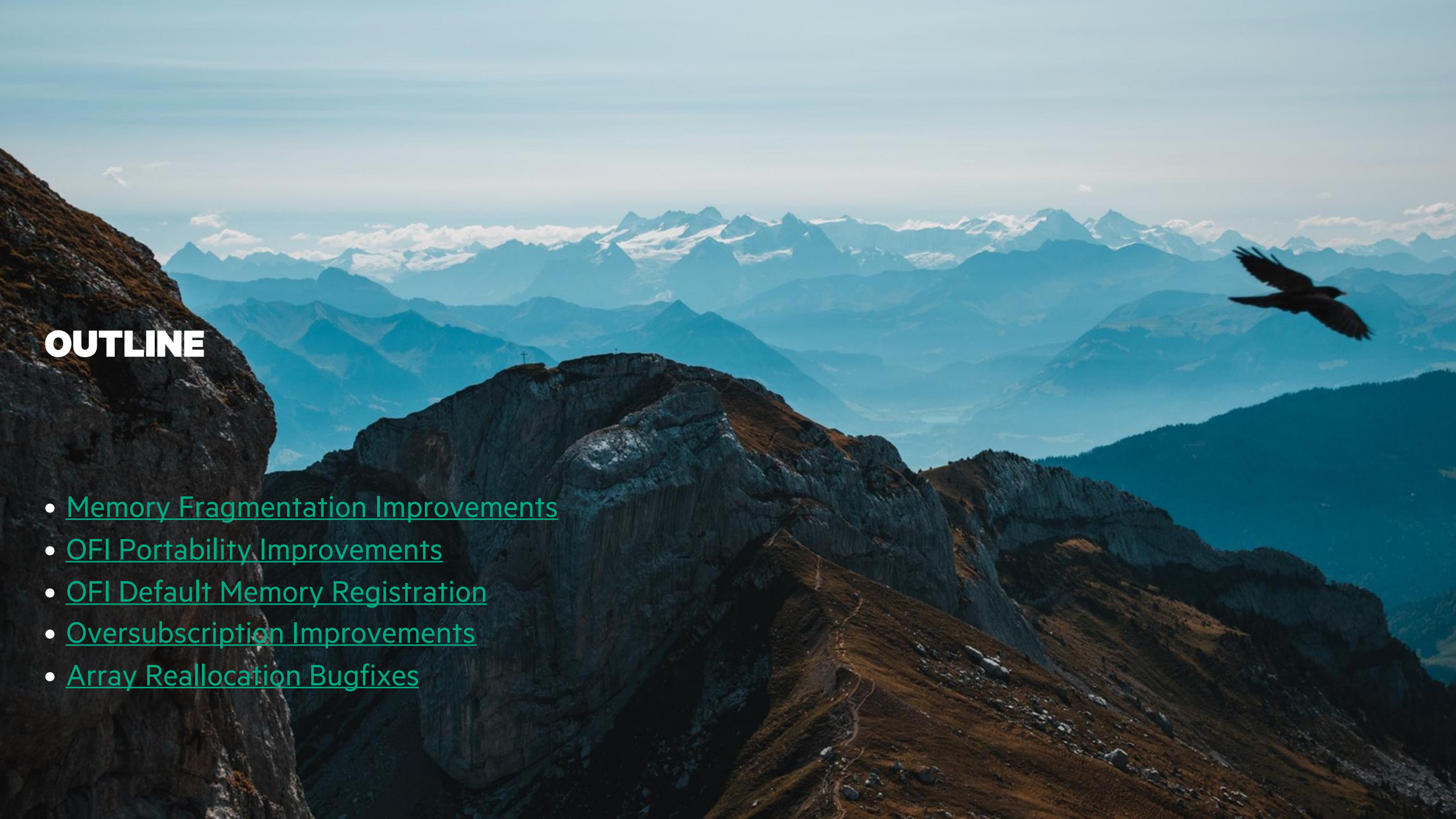
Hewlett Packard Enterprise

CHAPEL 1.25 RELEASE NOTES: RUNTIME IMPROVEMENTS

Chapel Team

September 23, 2021



OUTLINE

- Memory Fragmentation Improvements
- OFI Portability Improvements
- OFI Default Memory Registration
- Oversubscription Improvements
- Array Reallocation Bugfixes


MEMORY FRAGMENTATION
IMPROVEMENTS

MEMORY ALLOCATION
Background

- Arkouda users reported running out of memory on InfiniBand systems, despite few live allocations
  - In contrast to most HPC applications, Arkouda has highly dynamic allocation patterns
    - Allocates and frees many differently-sized large arrays during normal operation
  - This difference highlighted memory fragmentation issues we were previously unaware of
- Our memory allocation implementation is network-dependent
  - Memory must be registered with most HPC networks in order to do one-sided GETs/PUTs (RDMA)
  - How we satisfy user application allocations varies based on what is best for a particular network
- We use ‘jemalloc’ as our optimized memory allocator
  - It is a “general purpose malloc that emphasizes fragmentation avoidance and scalable concurrency support”
  - There are “chunk hooks” to manage lifetimes of memory chunks/regions (think ‘sbrk’/’mmap’ replacements)
    - This allows us to, for instance, have jemalloc satisfy allocations out of a registered chunk of memory

MEMORY ALLOCATION MODES
Background

- Today, we have 3 different memory allocation modes:
  - **simple**: comm none, gasnet-everything, ofi-tcp
    - Just use jemalloc as a replacement for the system allocator, any registration happens at comm time in the comm layer
    - Fragmentation is handled completely by jemalloc, can take advantage of virtual memory and use mmap for large arrays
  - **dynamic-heap**: ugni
    - Large arrays are separately allocated and registered through the comm layer at allocation time
      - Anything else is done with dynamic extensions to the heap via jemalloc chunk hooks
    - Fragmentation for large arrays is handled by the kernel, also taking advantage of virtual memory
  - **fixed-heap**: gasnet-ibv, ofi-verbs
    - We get a fixed segment or region of communicable memory from the comm layer
      - All allocations are satisfied out of this region via jemalloc chunk hooks
    - Fragmentation is still handled by jemalloc, but can only use provided memory region and not entire virtual address range

GASNET-IBV BACKGROUND

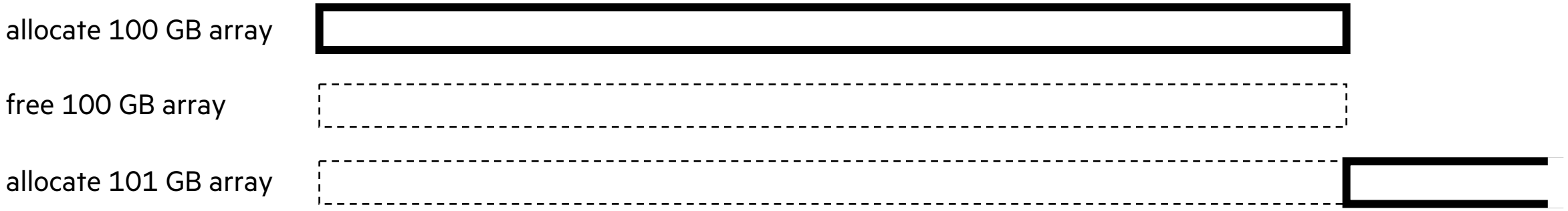
Background

- Chapel uses the GASNet communication library to target InfiniBand networks (gasnet-ibv)
  - gasnet-ibv runs with a fixed heap and has two different memory registration modes:
    - Static: All memory is registered at startup – fast communication, but hurts NUMA affinity and leads to long startup times
    - Dynamic: Memory is registered at communication time – can add overhead, but good NUMA affinity and startup times

- For configurations that require a fixed heap, we provide a jemalloc allocation chunk hook
  - This hook has an 'sbrk'-like like interface that only allows bumping an offset into the contiguous fixed heap
  - Did not provide merge/split hooks, which allow jemalloc to split or merge old chunks to satisfy new allocations
    - Not providing these led to severe fragmentation

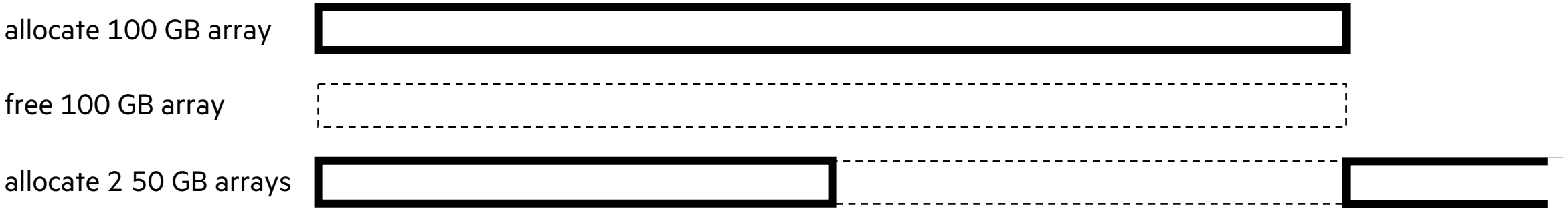
MERGE HOOK
Background

- Not providing merge hook led to fragmentation when using progressively larger allocations
  - e.g., an old 100GB allocation could not be used to help satisfy a new 101GB allocation



SPLIT HOOK
Background

- Not providing a split hook led to fragmentation when using progressively smaller allocations
  - e.g., an old 100GB allocation could not be split for 2 50GB allocations



MEMORY FRAGMENTATION

This Effort and Impact

This Effort: Implemented jemalloc merge and split hooks to reduce fragmentation

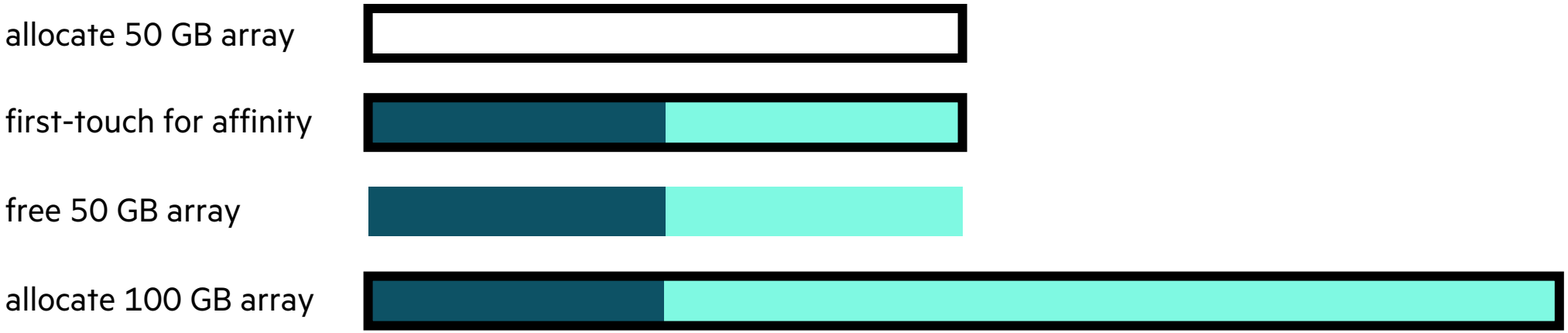
Impact: Increased amount of usable memory by 2-3x for some Arkouda workloads

- Results are comparable to configurations that use a dynamic heap (ugni)
- Situation is better, but not ideal
  - A small live allocation between two dead ones will prevent merging
  - Chunks are only merged after allocation, so large allocations do not try to extend old allocations proactively
- Ultimately, still trying to satisfy allocations out of a limited contiguous region of memory
  - Not taking advantage of massive virtual address space

MEMORY FRAGMENTATION

Performance Impact

- NUMA affinity, and thus performance was hurt under dynamic registration
  - Initial NUMA affinity is set by first-touch, but reusing previously touched memory results in suboptimal affinity



NUMA AFFINITY

This Effort

- As a short-term fix, added an --interleave-memory option to interleave pages on allocation
  - Similar to what we do for static registration, but done at allocation time instead of program startup

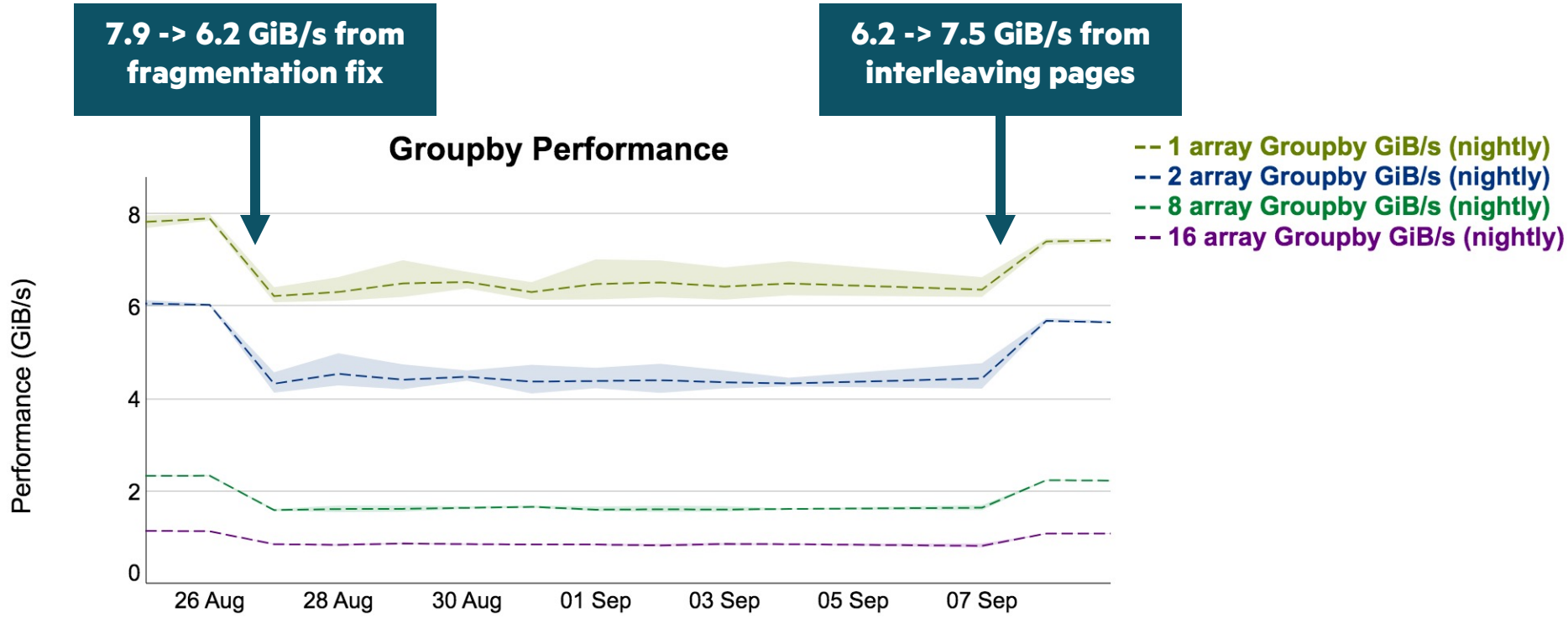
allocate 50 GB array

free 50 GB array

allocate 100 GB array

NUMA AFFINITY
Impact

- NUMA affinity is still suboptimal, but memory accesses are evenly split between NUMA domains
  - Overall, this stabilizes and improves Arkouda performance compared to memory reuse with first-touch
    - But still behind ideal first-touch (for this reason --interleave-memory is not the default, since it hurts traditional HPC codes)



MEMORY FRAGMENTATION

Next Steps

- Try implementing dynamic-heap style memory allocation
  - Will require collaborating with GASNet team to develop a registration/deregistration API
  - The simple implementation will result in always getting new pages from the system and registering
    - Relatively cheap under Aries, but we have seen that registration is slower under InfiniBand
    - Can explore using different chunk hooks for arrays so jemalloc can reuse similar sized allocations

- Explore running with a process per NUMA domain instead of process per node
  - Would permit using dynamic registration without having to worry about getting correct first-touch


OFI PORTABILITY IMPROVEMENTS

OFI PORTABILITY IMPROVEMENTS

Background and This Effort

Background:

- OFI communication layer is based on libfabric, defined by the Open Fabrics Interfaces (OFI) Working Group
  - Native network interface on HPE Cray EX systems, and is portable to others such as AWS/EFA
  - Defines an interface to an abstract network
  - Application selects a *provider* that instantiates the network with desired characteristics

This Effort:

- Upgraded bundled libfabric from v1.10.1 to v1.12.1.
- Added more extensive filtering to ignore undesirable providers returned by libfabric
  - Ignore 'sockets' provider that uses IPv6 addresses; doesn't work
  - Ignore 'verbs' provider that uses AF_IB addresses; doesn't work
  - Ignore providers that use the Mac T2 Security Chip interface
  - 'Sockets' provider is deprecated; warn if we use it
    - Only selected if specified via 'CHPL_RT_COMM_OFI_PROVIDER' or 'FI_PROVIDER' environment variables

OFI PORTABILITY IMPROVEMENTS

Impact and Next Steps

Impact:

- Greater portability with less special-case configuration

Next Steps:

- Continue to track libfabric upgrades and evolving provider capabilities
- Add more nightly testing


OFI DEFAULT MEMORY
REGISTRATION

OFI DEFAULT MEMORY REGISTRATION

Background:

- Many higher-performing providers (verbs, for example) require explicitly-registered memory
- We didn't default to a registered heap (which must be large to be useful) due to concerns about start-up time
- Thus, users on systems with high-performance networks got poor performance, from less-capable providers
- They had to set an environment variable to size a registered heap and get a higher-performing provider

This Effort:

- Bias the other way: Default to a large registered heap if that enables a higher-performing provider such as verbs
- Also register the static data segment, process heap, and process stack

Impact:

- Better default performance
- Longer start-up times

Next Steps:

- Investigate dynamic registration, as in CHPL_COMM=ugni, to reduce start-up time and improve NUMA affinity

OVERSUBSCRIPTION IMPROVEMENTS

OVERSUBSCRIPTION IMPROVEMENTS

Background and This Effort

Background:

- A locale does not have exclusive use of a node when oversubscribed
  - Failing to take this into account can lead to resource exhaustion
- Oversubscription typically happens during development and internal testing

This Effort:

- Automatically detect multiple locales on the same node for ‘ofi’ and ‘ugni’ comm layers
- Automatically size fixed heap
  - Allows use of libfabric providers that require fixed heap due to memory registration

OVERSUBSCRIPTION IMPROVEMENTS

Impact and Next Steps

Impact:

- Avoids memory exhaustion when oversubscribed

Next Steps:

- Add GASNet-EX functionality when Chapel supports it
- Investigate adjustments to tasking layer to share cores better when oversubscribed


ARRAY REALLOCATION BUGFIXES

ARRAY REALLOCATION BUGFIXES

Background: Since Chapel 1.22, some 1D rectangular arrays can be resized with realloc

- Before 1.22, this was always done with an alloc, copy, and free
- This was primarily motivated by single-node benchmarks that dynamically grow arrays
- A user reported crashes when resizing arrays on a Cray XC (comm=ugni)
- We use a dynamically extended heap under ugni, which requires a different allocation scheme than other modes
  - Large arrays are allocated through the comm layer for NUMA affinity reasons
  - Small arrays and everything else are allocated through the memory layer via dynamic heap extensions

This Effort: Fixed memory reallocation bugs under 'ugni'

- Fixed a bug in heap extension hooks that incorrectly told the allocator a reallocation was done in place
- Stopped using realloc for resizes that cross the threshold between a normal and comm-allocated array
  - We now do a full alloc, copy, free for that resize to avoid losing track of which allocator owns the memory

OTHER RUNTIME IMPROVEMENTS

OTHER RUNTIME IMPROVEMENTS

For a more complete list of runtime changes and improvements in the 1.25 release, refer to the following sections in the CHANGES.md file:

- ‘Generated Executable Flags’
- ‘Runtime Library Changes’
- ‘Bug Fixes’
- ‘Platform-specific Bug Fixes’
- ‘Third-Party Software Changes’
- ‘Developer-oriented changes: Runtime improvements’


THANK YOU
https://chapel-lang.org
@ChapelLanguage