RECENT INFINIBAND OPTIMIZATIONS IN CHAPEL

Elliot Ronaghan

CHIUW 2021 – June 4, 2021
Recent optimizations have significantly improved Chapel’s performance and scalability on InfiniBand:

- Approximately 15x performance improvement for Arkouda Argsort at 240 nodes (~30K cores)
INFINIBAND BACKGROUND

- Historically, the Chapel team primarily focused on performance for Cray networks
  - Intent was to ensure Chapel had the right language features/semantics first, then optimize for other networks

- More recently, focus has shifted to improving performance for InfiniBand networks
  - Chapel uses the GASNet communication library with the ibv conduit to target InfiniBand (gasnet-ibv)
Memory must be registered with the network in order to do one-sided GETs/PUTs (RDMA)

- gasnet-ibv supports two 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 fast startup

Chapel defaults to dynamic registration to get good NUMA affinity and fast startup times

- We believe this is the right choice for most users getting started
  - Have recommended static registration to some users with certain communication-heavy idioms in the past
- Ideally, we just want to have one mode with no, or few, downsides

Late in the 1.24 release cycle, we identified root cause of some InfiniBand performance issues

- Somewhat improved NUMA affinity and startup times for static registration (not covered in this talk)
- Significantly improved communication performance for dynamic registration (main topic for this talk)
  - These improvements motivated April’s 1.24.1 release
DYNAMIC REGISTRATION IMPROVEMENTS
DYNAMIC REGISTRATION BACKGROUND

- gasnet-ibv dynamic registration only registers memory at communication time
  - Fast startup time since little registration occurs at startup
  - NUMA affinity is based on user first-touch
  - Memory registration is expensive, want to amortize costs
    - Ideally only register a memory region once and then reuse
    - This requires tracking which memory regions are currently registered
DYNAMIC REGISTRATION IMPROVEMENTS

• Identified bottleneck in registration tracking code that limited performance and scalability
  • Core issue was that we were running out of dynamic registration entries
    – Led to deregistration and reregistration cycles, preventing amortization

• Collaborated with the GASNet team to resolve this issue
  • Increased number of dynamic registration entries based on execution-time query of hardware capabilities
  • Improved data structures used to track which regions are registered
SERIAL TRANSFER PERFORMANCE

- Significant performance improvements for codes with large point-to-point communication patterns
• Significant performance improvements for codes with all-to-all communication patterns
• Significant performance improvements for Arkouda
ARKOUDA BACKGROUND

• Arkouda provides NumPy-like arrays at HPC scale
  • A NumPy/Pandas Python interface, backed by Chapel
    - https://github.com/mhmerrill/arkouda

• We track Arkouda performance nightly at small-scale
  • Had an opportunity to run on a large HPE Apollo system
    - 128-cores – (2) 64-core AMD Rome Processors
    - 2 TB of memory
    - HDR-100 InfiniBand network
Previously, performance fell off above 64 nodes for Argsort
- Gather, Scatter, and other core idioms also suffered
Fixing dynamic registration improved performance and enabled tuning aggregation
- Argsort is ~50% faster at 16 nodes, ~15x faster at 240 nodes
CURRENT ARKOUDA SCALABILITY

• Fixing dynamic registration improved performance and enabled tuning aggregation
  • Gather and Scatter see similar improvements
ARGSORT BACKGROUND

• Argsort requires ~6x the input data for scratch space

• Previous Arkouda scalability graphs used 8 GiB-per-node input arrays
  • This was the largest power of 2 we could reliably sort on our XC
  • Apollo system has significantly more memory, allowing much larger problem sizes to be used

• Argsort has a fixed startup overhead that depends on \(\text{\textquoteleft numCores\texttimes numLocales\textquoteRight}\)
  • More cores on Apollo system means higher startup overhead, which can be amortized with larger problem sizes
Apollo system offers performance improvements over Cray XC, especially at larger problem sizes

- For equivalent problem sizes: ~2.5x improvement at 16 nodes and ~50% at 240 nodes
- For larger problem sizes: ~3x improvement at 16 and 240 nodes
Can run significantly larger problem sizes on Apollo system
  - Sorting up to 256 GiB per node input arrays (60 TiB at 240 nodes in under 5 minutes)

Arkouda Argsort Performance
chpl 1.24.1 / ak 04/06/21

- HPE Apollo -- 256 GiB Arrays
- HPE Apollo -- 64 GiB Arrays
- HPE Apollo -- 8 GiB Arrays
SUMMARY

• Performance and scalability of large transfers on InfiniBand systems has been improved
  • Dynamic registration communication performance is nearly on par with static registration
    – While retaining fast startup and good NUMA affinity
• Continue to improve dynamic registration performance
  • ISx and some other communication-intensive applications lag slightly still

• Look at using On-Demand-Paging (ODP) as an alternative registration mechanism
  • Hardware/firmware takes care of registration on-demand rather than tracking in software

• Improve other aspects of InfiniBand performance
  • Network injection is currently serialized, limiting performance of fine-grained communication
    – Mapping to the upcoming GASNet-EX multi-endpoint API should resolve this
  • Target the GASNet-EX network atomic API
THANK YOU

elliot.ronaghan@hpe.com