CHAPEL 1.25 RELEASE NOTES: PERFORMANCE OPTIMIZATIONS

Chapel Team
September 23, 2021
OUTLINE

- InfiniBand Optimizations
- Automatic Aggregation Improvements
- Barrier Optimizations
- Bounded Coforall Optimization Improvements
INFINIBAND OPTIMIZATIONS
Background

- 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
    - Have recommended static registration to some users with certain communication-heavy idioms
  - Ideally, we just want to have one mode with no, or few, downsides

- The 1.24.1 release included significant InfiniBand performance improvements
  - Many of these reduced the gap in communication performance between dynamic and static registration
  - For this release, we wanted to further tune performance and hopefully work towards a single registration mode
INFINIBAND OPTIMIZATIONS

Background and This Effort

**Background:** Discovered IB verbs completion queues (CQ) were being highly contended
- CQs are polled to track the completion of network operations
- Currently, multiple threads share a single CQ, which leads to concurrent polling and contention
- CQs are protected by an unaligned lock in the verbs API
- Unaligned lock led to false-sharing, which compounded performance penalty
  - Bottleneck was identified with perf-c2c, a tool that helps identify cacheline contention

**This Effort:**
- Collaborated with GASNet team to serialize CQ polling with an aligned try-lock
  - Try-lock skips polling if the lock is already held, reducing total number of polling calls and contention
  - Alignment eliminates false-sharing
INFINIBAND OPTIMIZATIONS

Impact

- Significant performance improvements for applications with concurrent communication

-NPB: FT Perf (Mflops/s) - size D

-35% improvement for static registration
INFINIBAND OPTIMIZATIONS

Impact

• Significant performance improvements for applications with concurrent communication

---

Bale: Indexgather Perf (MB/s per node)

- ~2x improvement for static registration
- ~25% improvement for dynamic registration
INFINIBAND OPTIMIZATIONS

Impact

- Significant performance improvements for Arkouda with dynamic registration
INFINIBAND OPTIMIZATIONS

Impact

- Significant reduction in variability on systems with Address Space Layout Randomization (ASLR)
  - ASLR led to randomized CQ lock addresses, which made the impact of false-sharing variable from run-to-run
  - Our test systems run with ASLR disabled, but many sites have it enabled
    - GASNet results show improved stability on systems with ASLR (L: linear-scale 64-core Intel, R: log-scale 128-core AMD)
INFINIBAND OPTIMIZATIONS

Next Steps

• Further reduce CQ contention by using the GASNet-EX multi-endpoint API
  • Creating an endpoint and CQ per thread can reduce contention

• Improve dynamic registration performance
  • CQ polling optimizations widened the gap between dynamic and static registration performance

• 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
  • Current prototype hangs
    – Needs more investigation and collaboration with the GASNet team

• Gather performance comparisons between Chapel and reference MPI/SHMEM codes
  • Use this to drive further optimizations
AUTOMATIC AGGREGATION IMPROVEMENTS
AUTOMATIC AGGREGATION IMPROVEMENTS
Background and This Effort

Background:
• In Chapel 1.24, we added a compiler optimization to aggregate remote communication

```chapel
forall i in A.domain do
    A[i] = B[computeIndex(i)];    // accesses to B are aggregated
```

• The optimization is off-by-default and can be enabled with ‘--auto-aggregation’

This Effort:
• More comprehensive coverage for automatic aggregation
• Performance improvements
AUTOMATIC AGGREGATION IMPROVEMENTS
Impact – Improved Coverage

- Local, non-distributed arrays are recognized as local

```plaintext
var A = newBlockArr(1..10, int);
coforall l in Locales do on l {
    var localArr: [1..10] int;
    forall i in localArr.domain do
        localArr[i] = A[computeIndex(i)];
}
```

Access to a local array that’s declared on ‘here’

Communication will be aggregated
AUTOMATIC AGGREGATION IMPROVEMENTS
Impact – Improved Coverage

• Local, non-distributed arrays are recognized as local

```
var A = newBlockArr(1..10, int);
coforall l in Locales do on l {
    var localArr: [1..10] int;
    forall i in localArr.domain do
        localArr[i] = A[computeIndex(i)];
}
```

Access to a local array that’s declared on ‘here’
Communication will be aggregated

• Explicit calls to ‘localAccess’ recognized as local

```
var A = newBlockArr(1..10, int);
var B = newBlockArr(1..10, int);
forall i in A.domain do
    A.localAccess[computeLocalIndex(i)] = B[computeIndex(i)];
```

Left-hand side is local because of ‘localAccess’
Communication will be aggregated
AUTOMATIC AGGREGATION IMPROVEMENTS
Impact – Improved Performance

• Changes made to improving aggregation in Arkouda were incorporated in upstream Chapel

Around 20% improvement for automatic and manual aggregation
AUTOMATIC AGGREGATION IMPROVEMENTS

Next Steps

• Provide user-facing aggregation (#16963)

• Port more bale apps for testing aggregation

• Improve all-local aggregation performance

• Extend the coverage to promoted expressions

• Investigate multi-hop aggregators for better memory scalability
BARRIER OPTIMIZATIONS
**BARRIER OPTIMIZATIONS**

**Background and This Effort**

**Background:** At CHIUW 2021, the CHAMPS team reported performance issues in synchronization code

- Synchronization is implemented with a variant of the ‘allLocalesBarrier’
- Discovered excessive communication on every ‘barrier’ call
  - Due to the implementation using a distributed array in a class, which is a known performance issue (#10160)

**This Effort:** Optimized ‘allLocalesBarrier’

- Moved distributed array out of the class to eliminate all communication beyond the inter-node barrier itself
  - Workaround until performance issues around distributed array fields are resolved
**BARRIER OPTIMIZATIONS**

**Impact**

- Significantly faster barrier, especially for configurations where concurrent communication is slow
  - On 16 nodes of a Cray CS with InfiniBand, barrier is roughly 14x faster

![Graph showing performance comparison between InfiniBand and Aries.]
**BARRIER OPTIMIZATIONS**

**Impact**

- Significantly faster barrier at scale, even for configurations where concurrent communication is fast
  - On 512 nodes of a Cray XC with Aries, barrier is roughly 18x faster

![Graph showing barrier times for 512-node allLocalesBarrier (100,000 trials)]
BOUNDDED COFORALL
OPTIMIZATION IMPROVEMENTS
BOUNDDED COFORALL OPTIMIZATION IMPROVEMENTS

Background and This Effort

**Background:** Chapel 1.15 added a bounded coforall optimization
- Reduces task-tracking overhead for coforalls with a known trip-count (ranges, domains, arrays)
- During 1.25 we discovered this optimization did not fire for zippered coforalls
  - Identified while optimizing ‘Block’ array creation communication
  - Used zippered iteration to reduce communication, but execution time suffered due to slower task-tracking

**This Effort:** Extended the bounded coforall optimization to include zippered iteration
- This enabled optimizing communication for BlockDist array creation
**BOUNDED COFORALL OPTIMIZATION IMPROVEMENTS**

**Impact**

- Performance improvements for codes using bounded zippered coforalls

![Diagram showing LLNL CoMD Time (sec) with a ~15% faster force computation annotation](image-url)
Impact

• Communication count reduction for BlockDist array creation
OTHER PERFORMANCE IMPROVEMENTS
OTHER PERFORMANCE IMPROVEMENTS

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

- ‘Performance Optimizations/Improvements’
THANK YOU

https://chapel-lang.org
@ChapelLanguage