Richard Jefferson
CSCI 415

5/31/2025

Assignment 5: Sorting in Chapel

1. Is Chapel Competitive with C's Sequential Implementation?

Based on the measured runtimes, Chapel is not competitive with C's `qsort()` in sequential or small-scale scenarios. The C implementation of `qsort()` consistently outperformed both Chapel versions by large margins, even at larger array sizes. For example, `qsort()` sorted over a million elements in just 0.18 seconds, while Chapel's hybrid sort took over 43 seconds, and the bitonic sort exceeded 13 minutes. This stark contrast highlights Chapel's significant overhead and focus on scalability rather than minimal runtime.

However, Chapel becomes more appealing when scaling across multiple locales. The hybrid version takes advantage of local quicksorts for fast intra-locale performance and complements it with a parallel bitonic merge. While still slower than C for the sizes tested, the hybrid strategy demonstrates much better efficiency than bitonic alone.

Conclusion: C's `qsort()` is far superior for small to moderate datasets on single nodes. Chapel's strengths only begin to surface on large-scale distributed workloads, particularly when using hybrid techniques.

2. Which Chapel Program Is Fastest and in What Circumstances?

Program	Best Use Case	Strengths	Weaknesses
bitonic.chpl	Very large datasets across many locales	High parallelism, simple structure	Very poor performance for medium datasets
hybrid.chpl	Medium-to-large arrays, 4–16 locales	Balances fast local sort with scalable merge	Requires more careful data management

The hybrid Chapel program consistently outperformed the pure bitonic sort. At 1 million elements, the hybrid sort completed in 43.72 seconds compared to 819.53 seconds for bitonic. This ~19x improvement highlights the advantage of using quicksort locally. The gap remained significant across all tested sizes.

Bitonic sort does offer conceptual simplicity and uniform work distribution, but its rigid recursive merging does not scale efficiently. Hybrid sorting strikes a much better balance

and should be the default choice when performance matters.

Conclusion: Use `hybrid.chpl` for practical performance. Only consider `bitonic.chpl` in educational or ultra-large domain experiments.

3. Evaluation of Chapel as a Programming Language

Chapel simplifies parallel and distributed programming through high-level abstractions. Constructs like `forall`, `on`, and `dmapped` allow developers to express parallelism and locality with minimal boilerplate. This makes it easier to implement scalable algorithms without diving into MPI or thread management.

However, Chapel still suffers from significant runtime overhead and limited optimization compared to C. Its performance can lag behind unless used in highly parallel, multi-locale environments. The ecosystem is also relatively young, with fewer libraries and community resources available.

Conclusion: Chapel is an effective research and education tool for distributed computing, but it's not a direct competitor to C for high-performance serial tasks.

4. Timing Table

Array Size	Locales	quicksort.c Time	bitonic.chpl Time	hybrid.chpl Time
2^{14} = 16,384	1	0.0019 sec	9.60 sec	0.78 sec
2^{16} = 65,536	4	0.0089 sec	42.50 sec	3.04 sec
2^{17} = 131,072	4	0.0201 sec	94.05 sec	6.45 sec
2^{18} = 262,144	4	0.0431 sec	173.17 sec	10.19 sec
2^{20} = 1,048,576	4	0.1813 sec	819.53 sec	43.72 sec