Scalable Load-Balancing Concurrent Queues on Many-core Architectures

Caleb Lehman[1], Poornima Nookala[2] (advisor), Ioan Raicu[2] (advisor)
[1]Ohio State University, [2]Illinois Institute of Technology

Abstract

- Core counts are increasing, making parallel programming an increasingly powerful paradigm
- Decomposition of computations into very fine-grained tasks is necessary to effectively utilize many-core systems
- We present **XQueue**, a novel design for a queuing system, analyze its efficiency, suggest improvements, and compare performance to existing designs, all within the context of **XTask**, a custom, task-based runtime for shared memory systems.

Motivation

- Swift/T is an implicitly parallel programming language used to implement scientific dataflow programs.
- Swift/T uses MPI for internode and intranode communication.
- Today's processors can run billions of instructions per second, however we are limited to 100K tasks per second.
- Traditional concurrent data structures and synchronization mechanisms do not scale to hundreds of cores.

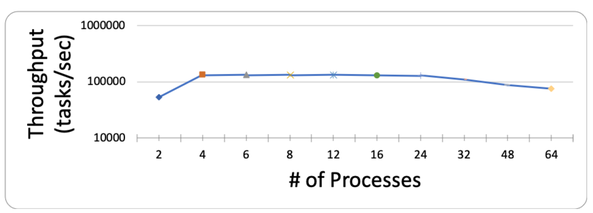
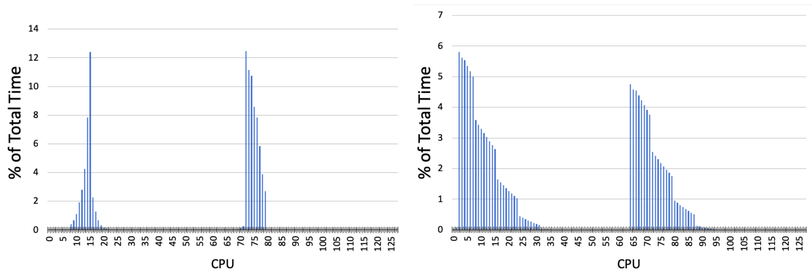


Figure 1: Throughput of Swift/T in tasks/sec for Fibonacci workload on a Haswell machine with 24 cores and 48 hardware threads.

Can high-level programming be applied to modern parallel architectures with strong scaling workloads?

Load Balancing in XQueue

The initial **XQueue** design provides lower latencies for individual enqueue and dequeue operations, but has very poor load balancing, leading to relatively poor performance on our sample workloads.



These figures represent the number of cycles each CPU spent executing tasks (measure of load balancing) for the Fibonacci workload (left) and Cholesky workload (right).

XQueue Design

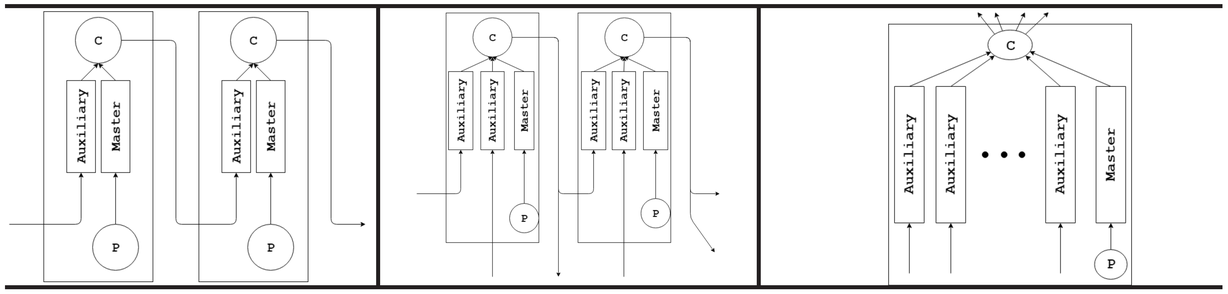


Figure 2: From left to right, the **XQueue**, **XQueue v2**, and **XQueue vN** designs (showing 2, 2, and 1 cores, respectively). In general, consumer threads pass tasks to auxiliary queues of connected cores and producer threads put tasks on local master queue. Consumer threads retrieve from all local queues.

Performance Results

Load Balancing

Let c_i denote the number of cycles the ith CPU spent executing tasks. For n CPUs, define *load balance error* by

$$\lambda := \frac{\max\{c_i\}}{\sum c_i / n} - 1$$

Our measurements yielded the following results:

Design	λ Fib.	λ Chol.
XQueue	14.96	4.85
XQueue v2	3.99	7.94
XQueue vN	0.12	0.03
Work Stealing	0.10	0.01

Fibonacci Results

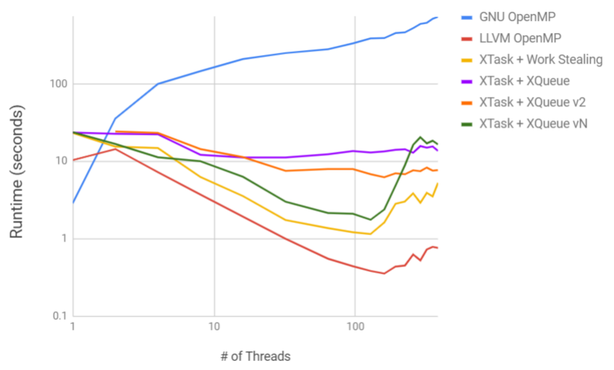


Figure 3: Runtimes of `Fib(35)` on Skylake 192-core machine with up to 384 hardware threads.

Cholesky Results

Figure 4: Performance on Cholesky workload (32K x 32K matrix) on AMD Epyc 64-core machine using 128 hardware threads.

Conclusions

- **XQueue** is a novel lockless concurrent queuing system with relaxed ordering semantics that is geared to realizing scalability to hundreds of concurrent threads.
- Original **XQueue** design had poor load balancing and depended significantly on workload structure.
- New designs significantly improved load balancing and slightly improved performance, but may have introduced some scalability concerns.
- The various **XQueue** designs performed worse than the standard work stealing approach.

Future Work

- Profile the **XTask** runtime to determine if it is optimized enough for testing **XQueue**
- Integrate **XQueue** into existing runtime systems
- Study new designs on same or larger systems

Acknowledgements

This work is supported by the NSF CCF-1757964/1757970 REU award (BigDataX), and the NSF CNS-1730689 CRI award (Mystic)