Shared Stack Implementation and Analysis Using Queue Delegation with Elimination (Final Project)

Esha Choukse (ec27876), Mike Thomson (mt29253)
Multicore Computing, Fall 2014

November 20, 2014

Abstract

Queue delegation and Elimination are two techniques that help us improve the performance of shared data structures, by reducing contention. We use both these concepts together to implement a shared stack and analyse the performance we get from it. We have used the C++ libraries published by Klaftenegger [**?**], and implemented elimination on top of it. We compare the performance of our implementation against the basic queue delegation, and MonitorT [**?**].

Shared data structures between threads, always tend to slow down the execution due to the inherent sequentiality associated with them. A lot of implementations still use locks. Locks, because of their mutual exclusivity, make the execution of the critical sectoin completely sequential. Several techniques are used to alleviate this disadvantage. One of these techniques is Queue delegation. Queue delegation allows a thread to offload its task to a delegate thread, which is already operating on the shared structure. This allows the thread to carry on with its execution, till the delegate thread executes the queued task. Another technique is to use Elimination. Elimination tries to take advantage of operations on the shared data structure, which have reverse semantics, for example a push and a pop on a stack. This way, none of the two threads actually access the shared data structure, resulting in less contention. We use both these concepts together to implement a shared stack and analyse the performance we get from it. We have used the C++ libraries published by Klaftenegger, and implemented elimination on top of it. We compare the performance of our implementation against the basic queue delegation, and MonitorT.

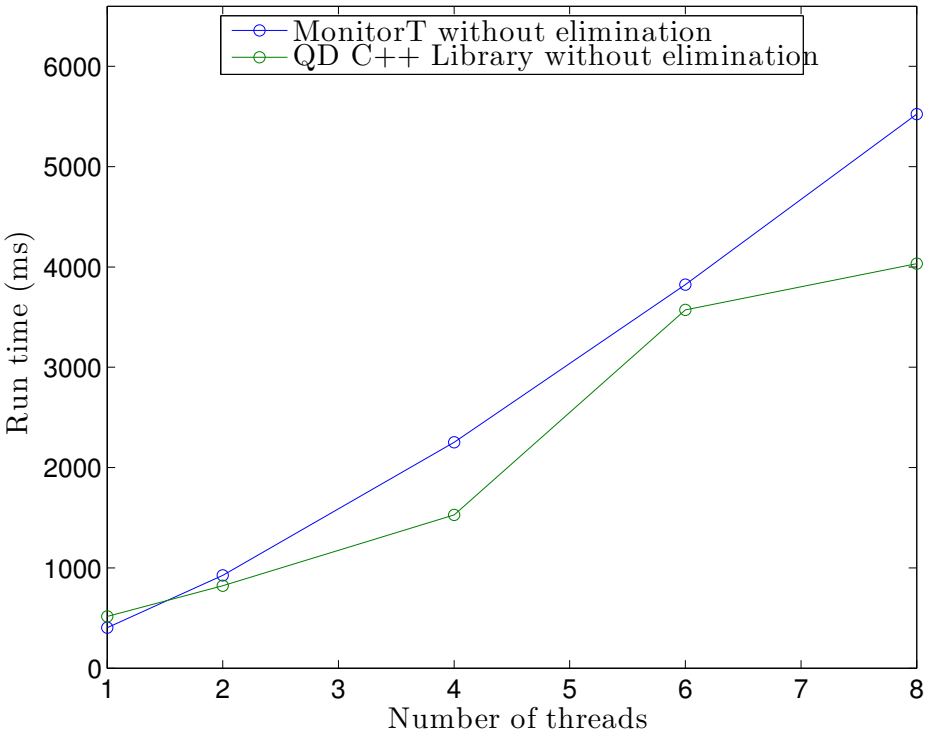


Figure 1: Baseline comparison of the MonitorT implementation (Java) and the QD Lock (C++), varying the number of threads, and without elimination for either implementation.

1 Results

2 Analysis

3 Conclusion

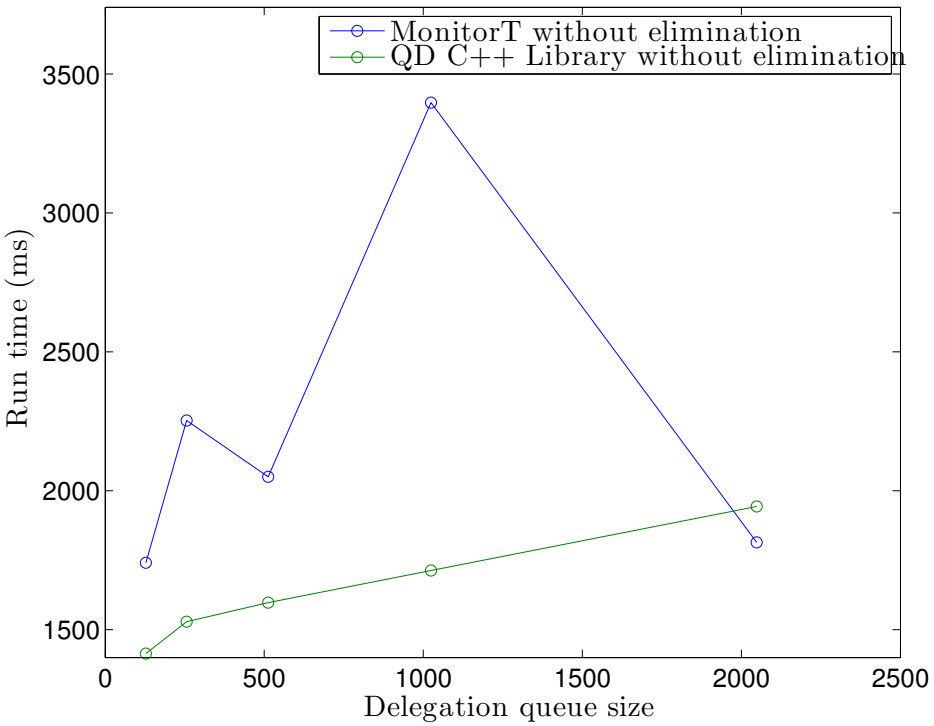


Figure 2: Baseline comparison of the MonitorT implementation (Java) and the QD Lock (C++), varying the delegation queue size, and without elimination for either implementation.

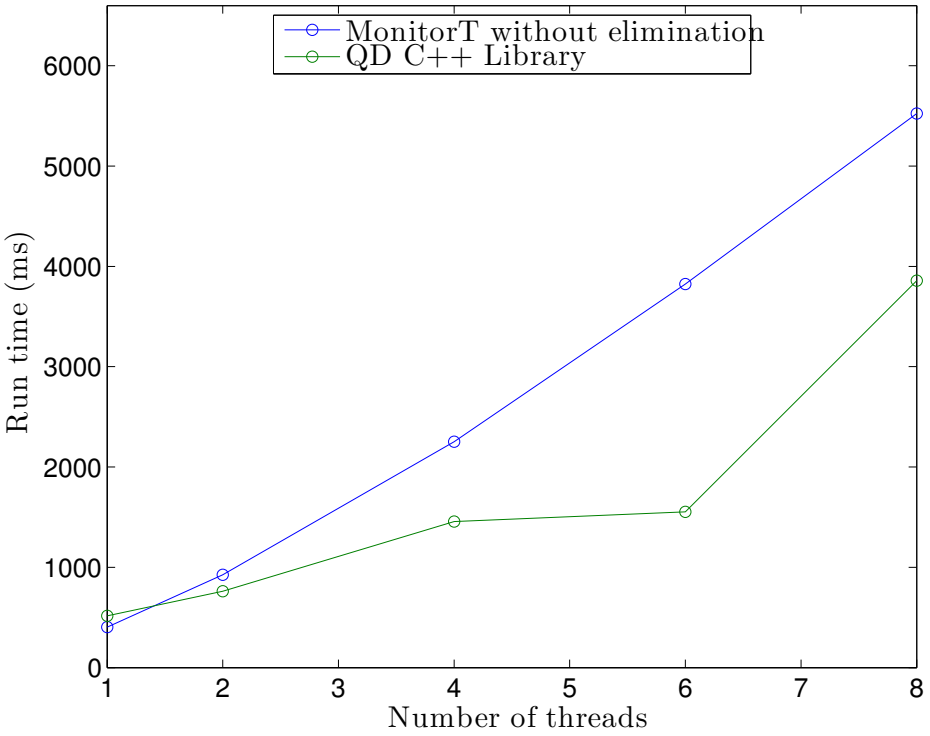


Figure 3: Comparison of the MonitorT implementation (Java) and the QD Elimination Lock (C++), varying the number of threads (elimination included for C++ implementation, not for Java). All future analysis will focus on the benefit of QD Elimination Lock over QD Lock (only C++ implementations).

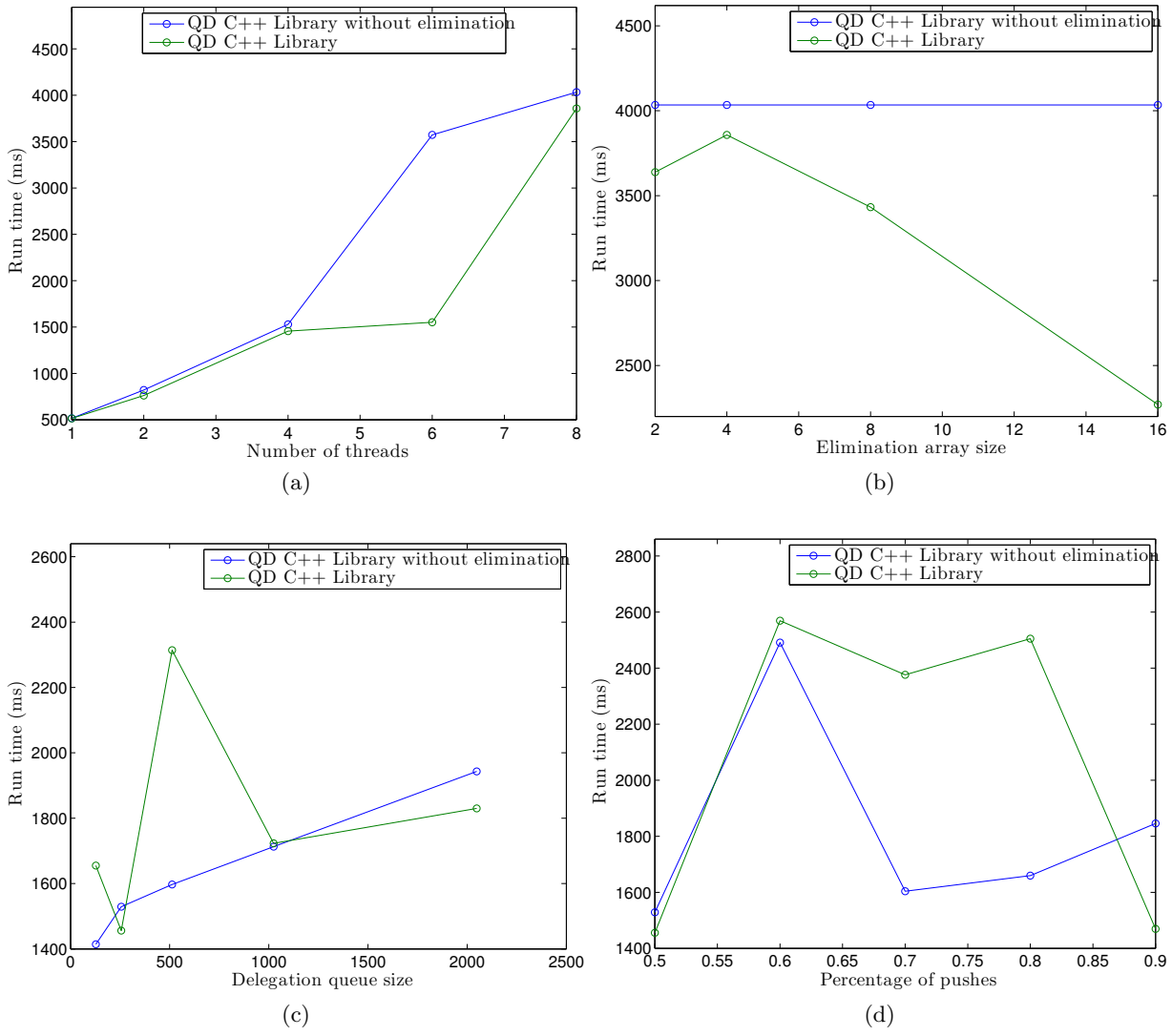


Figure 4: caption: **??** caption; **??** caption; **??** caption; **??** caption.

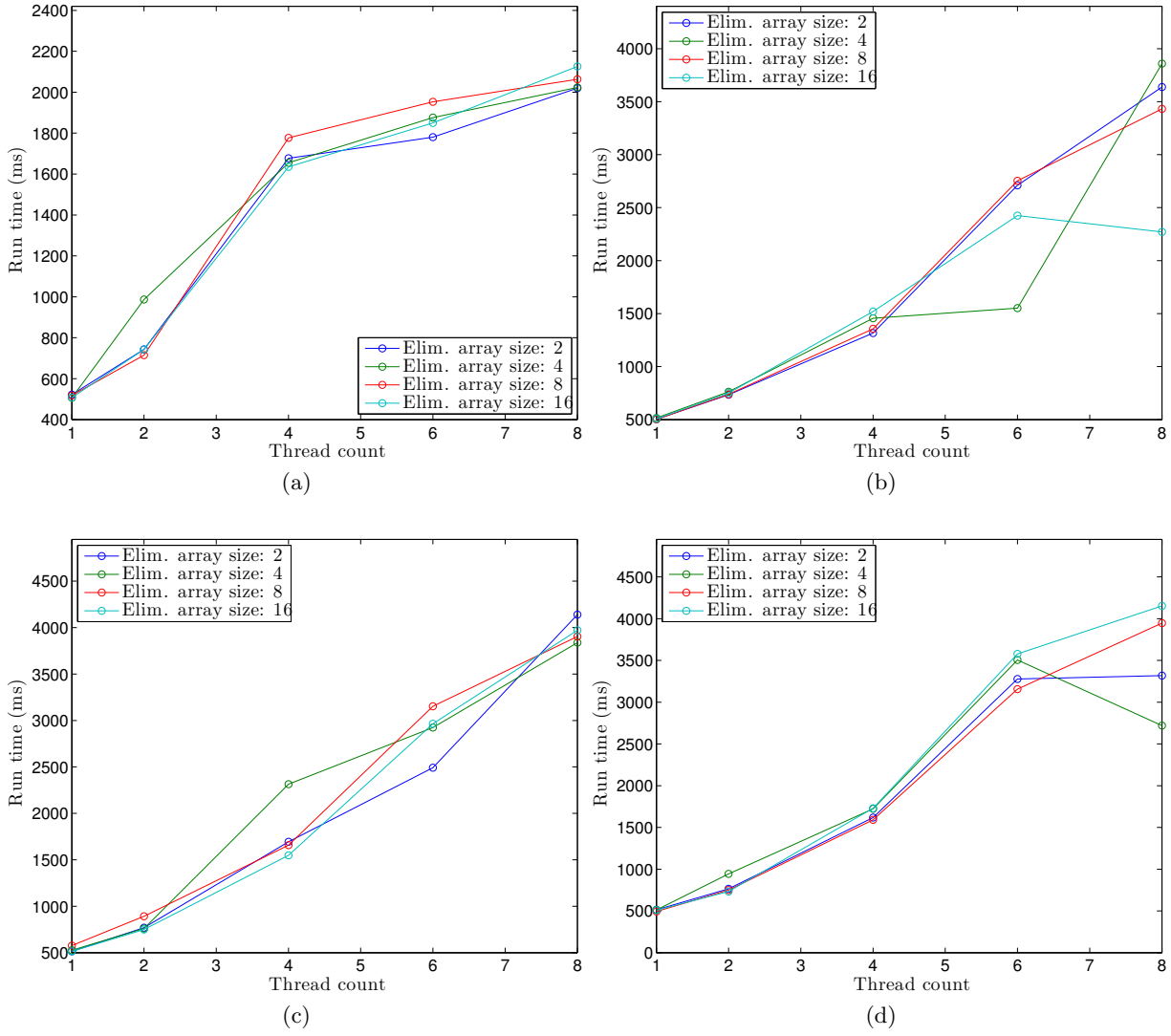
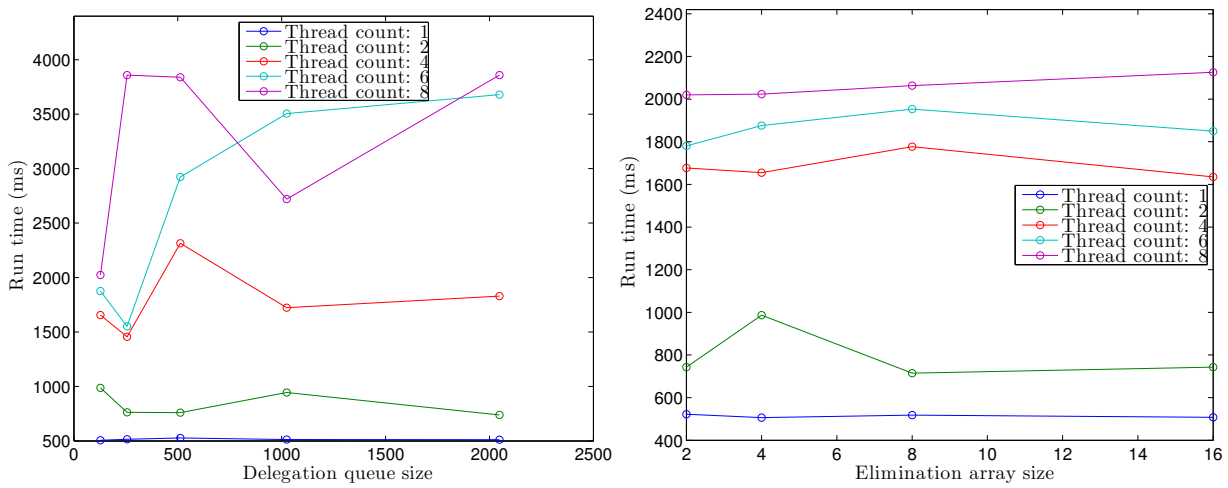


Figure 5: caption: ?? caption; ?? caption; ?? caption; ?? caption.



(a) (b)

Figure 6: caption: **??** caption; **??** caption.