Local Linearizability

Ana Sokolova

joint work with:

Andreas Haas
Andreas Holzer
Michael Lippautz
Ali Sezgin

Tom Henzinger
Christoph Kirsch
Hannes Payer
Helmut Veith
Semantics of concurrent data structures

- **Sequential specification** = set of legal sequences
  
  e.g. pools, queues, stacks

  e.g. queue legal sequence
  enq(1)enq(2)deq(1)deq(2)

- **Consistency condition** = e.g. linearizability / sequential consistency

  e.g. linearizable queue concurrent history

  t1: enq(2) deq(1)
  t2: enq(1) deq(2)
Consistency conditions

Linearizability [Herlihy,Wing ’90]

there exists a sequential witness that preserves precedence

Sequential Consistency [Lamport’79]

there exists a sequential witness that preserves per-thread precedence (program order)
Performance and scalability

![Graph showing throughput vs. number of threads/cores with different line styles and emoticons]
Relaxations allow trading correctness for performance
Relaxing the Semantics

- **Sequential specification** = set of legal sequences
- **Consistency condition** = e.g. linearizability / sequential consistency

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL'13

Local linearizability in this talk

not “sequentially correct”

for queues only (feel free to ask for more)

too weak
Local Linearizability
main idea

- **Partition** a history into a set of local histories
- **Require** linearizability per local history

Already present in some shared-memory consistency conditions (not in our form of choice)

Local sequential consistency… is also possible

no global witness
Local Linearizability (queue) example

(t1-induced history, linearizable)

(t2-induced history, linearizable)

(sequential) history not linearizable

locally linearizable
Local Linearizability (queue) definition

Queue signature $\Sigma = \{\text{enq}(x) \mid x \in V\} \cup \{\text{deq}(x) \mid x \in V\} \cup \{\text{deq}(\text{empty})\}$

For a history $h$ with $n$ threads, we put

$\text{In}_h(i) = \{\text{enq}(x)^i \in h \mid x \in V\}$

$\text{Out}_h(i) = \{\text{deq}(x)^i \in h \mid \text{enq}(x)^i \in \text{In}_h(i)\} \cup \{\text{deq}(\text{empty})\}$

$h$ is locally linearizable iff every thread-induced history $h_i = h \mid (\text{In}_h(i) \cup \text{Out}_h(i))$ is linearizable.
Generalizations of Local Linearizability

Signature $\Sigma$

For a history $h$ with $n$ threads, choose

- $\text{In}_h(i)$
- $\text{Out}_h(i)$

by increasing the in-methods, LL gradually moves to linearizability

- in-methods of thread $i$, methods that go in $h_i$
- out-methods of thread $i$, dependent methods on the methods in $\text{In}_h(i)$ (performed by any thread)

$h$ is locally linearizable iff every thread-induced history $h_i = h \upharpoonright (\text{In}_h(i) \cup \text{Out}_h(i))$ is linearizable.
Where do we stand?

In general

Local Linearizability

---

Linearizability

---

Sequential Consistency
Where do we stand?

For queues (and all pool-like data structures)

Local Linearizability

Linearizability

Sequential Consistency
Where do we stand?

Local Linearizability & Pool-seq.cons.

Linearizability

Sequential Consistency
Local linearizability is compositional

\( h \) (over multiple objects) is locally linearizable
iff
each per-object subhistory of \( h \) is locally linearizable

Local linearizability is modular / “decompositional”

uses decomposition into smaller histories, by definition

allows for modular verification

like linearizability
unlike sequential consistency
Verification (queue)

Queue sequential specification (axiomatic)

\[ s \text{ is a legal queue sequence} \]
\[ \text{iff} \]
\[ 1. \ s \text{ is a legal pool sequence, and} \]
\[ 2. \ \text{enq}(x) <_s \text{enq}(y) \land \text{deq}(y) \in s \Rightarrow \text{deq}(x) \in s \land \text{deq}(x) <_s \text{deq}(y) \]

Queue linearizability (axiomatic)

\[ h \text{ is queue linearizable} \]
\[ \text{iff} \]
\[ 1. \ h \text{ is pool linearizable, and} \]
\[ 2. \ \text{enq}(x) <_h \text{enq}(y) \land \text{deq}(y) \in h \Rightarrow \text{deq}(x) \in h \land \text{deq}(y) \not<_h \text{deq}(x) \]
Verification (queue)

Queue sequential specification (axiomatic)

\( s \) is a legal queue sequence
iff
1. \( s \) is a legal pool sequence, and
2. \( \text{enq}(x) \prec_s \text{enq}(y) \land \text{deq}(y) \in s \implies \text{deq}(x) \in s \land \text{deq}(x) \prec_s \text{deq}(y) \)

Queue local linearizability (axiomatic)

\( h \) is queue locally linearizable
iff
1. \( h \) is pool locally linearizable, and
2. \( \text{enq}(x) \prec_h \text{enq}(y) \land \text{deq}(y) \in h \implies \text{deq}(x) \in h \land \text{deq}(y) \prec_h \text{deq}(x) \)
Generic Implementations

Your favorite linearizable data structure implementation

LLD $\Phi$ pool linearizable & locally linearizable

local inserts / global (randomly distributed) removes

segment of dynamic size ($n$)

$\Phi$

$\Phi$

$\Phi$

$\Phi$
Performance

![Graph showing performance comparison between MS queue and LLD MS queue](image)

- **LLD MS queue** performs significantly better than **MS queue**

**Legend:**
- **LLD MS queue**
- **MS queue**

**Note:**
- The graph compares the performance of MS queue and LLD MS queue in terms of million operations per second (more is better) against the number of threads. LLD MS queue outperforms MS queue significantly.
Performance

![Graph showing performance against number of threads]

LLD $\Phi$ performs significantly better than $\Phi$
Performance

Thank You!

LLD MS queue performs better than the best known pools

MS queue
- LCRQ
- $k$-FIFO ($k=80$)

LL $k$-FIFO ($k=80$)
- static LL DQ ($p=40$)
- 1-RA DQ ($p=80$)

LLD MS queue