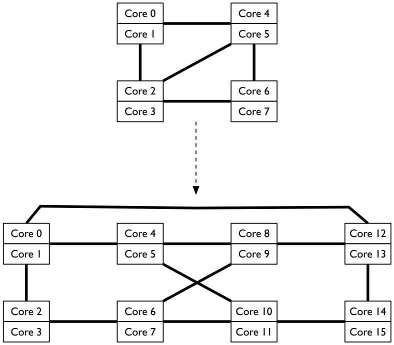
Concurrency Kit

Towards accessible non-blocking technology for C

Samy Al Bahra
AppNexus, Inc.
August 26, 2012

History



Core 0
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core 0
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core 8
Core 9
Core 10
Core 11
Core 12
Core 13
Core 14
Core 15

Motivation

Research and develop scalable synchronization methods for NUMA architectures.

What did we need?

- Concurrent memory model
- Support for atomic operations

Concurrency Primitives

Spinlocks

Algorithm	cas	dec_zero	faa	fas	load	store
anderson	◇		◇		◇	◇
cas	◇					◇
clh				◇	◇	◇
dec		◇			◇	◇
fas				◇		◇
ticket			◇		◇	◇
ticket_pb			◇		◇	◇
mcs	◇			◇	◇	◇

RW

Algorithm	cas	inc	dec	faa	fas	load	store	RMO
brlock	○				○	◇	◇	◇
bytelock	◇	◇	◇	○		◇	◇	◇
rwlock_naive	○	◇	◇		○	◇	◇	◇

Concurrency Primitives

Atomic Operation Support

Operation	apr	**atomic_ops**	**ck**	gcc	glib	liblfds	**urcu**
add			◇				◇
and			◇		◇		◇
btc			◇				
btr			◇				
bts			◇				
cas		◇	◇	◇		◇	
cas2		◇	◇				
cas2_value		◇	◇			◇	
cas_value	◇	◇	◇	◇	◇	◇	◇
dec	◇		◇				◇
dec_zero			◇		◇		
faa	◇	◇	◇		◇		◇
fas	◇	◇	◇	◇			◇
inc	◇		◇		◇	◇	◇
inc_zero			◇				
load	◇	◇	◇		◇		◇
neg			◇				
neg_zero			◇				
not			◇				
or		◇	◇		◇		◇
store	◇	◇	◇		◇		◇
sub	◇		◇				
xor			◇		◇		
Memory Model	G	R	R	G	G	N	R

Memory Models

Intel Architecture Manual, Volume 3A, 8.5

"Software should access semaphores (shared memory used for signalling between multiple processors) using identical addresses and operand lengths. For example, if one processor accesses a semaphore using a word access, other processors should not access the semaphore using a byte access."

- Is wait-free atomic snapshot with respect to loads and CAS2 possible?
- Implications on algorithms with asymmetric accesses.
- How can higher-level memory models accomodate ordering exceptions?

Bytelock read acquisition in the real world: 4 ticks → 38 ticks

appnexus

Concurrency Primitives

Memory Ordering Modifiers

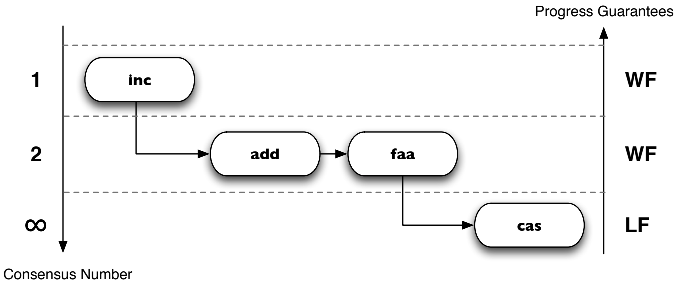
Library	LoadLoad			LoadStore			StoreLoad			StoreStore		
atomic_ops	○	◇	○	○	◇	○	○	◇	○	○	◇	○
ck	●	◇	◇							●	◇	◇
freebsd	◇	◇	★							◇	◇	★
linux	●	◇	◇							●	◇	◇
urcu	●	◇	◇							●	◇	◇
	x86-64	Power	SPARC	x86-64	Power	SPARC	x86-64	Power	SPARC	x86-64	Power	SPARC

● = NOP (strict fallback)
○ = NOP
★ = Full semantics
◇ = Yes

Concurrency Primitives

Portability

A port should require little besides atomic load, store, fences and a single universal atomic primitive.



Transparency

Non-blocking progress guarantees should reflect rank in the wait-free hierarchy.

Concurrency Primitives

The ABA Problem

- Algorithms subject to CAS-based speculation are prone to ABA.
- ABA-safety typically provided with ABA counters or SMR.
- Currently, we have pigeon-holed LL/SC-based architectures into SMR.

N1548

"The weak compare-and-exchange operations may fail spuriously, that is, return zero while leaving the value pointed to by expected unchanged."

↓

The operation is guaranteed to fail if object was the destination of any concurrent write operations, regardless of functional side-effects.

Concurrency Primitives

Atomics License

atomic_ops	**ck**	urcu
MIT	Revised BSD	LGPL2.1

LGPL2 is restrictive

"If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work ... Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself."

Summary of Opinions

- There is no silver bullet, concurrent applications that care about performance eventually require their own concurrent memory model
- A model should:
  - Expose wait-free hierarchy
  - Allow for exceptions to the memory model
- An implementation should optimize for underlying instruction set
- LL/SC needs love, ABA-safety yields fair number of performance wins

Safe Memory Reclamation

Safe Memory Reclamation

Examples of Progress

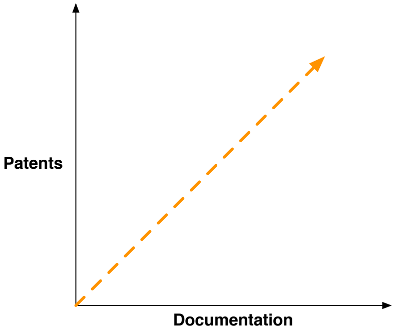
"Thank you for your message. I have no knowledge of the legal issues related to the use of [...], or to represent [...]'s position ..."

State of the World

EBR	HP	PTB	PC	QSBR
○	●	●	○	○

○ = Available (at least partially) ● = Locked

Safe Memory Reclamation



Literature Survey

- HP, RCU, QSBR and PTB have comprehensive documentation
- Proxy collectors backed by mailing list threads and one brief blog post
- A handful of paragraphs exploring epoch-based reclamation
    - Primarily thanks to Thomas Hart and Paul McKenney

appnexus

Safe Memory Reclamation

Bringing hobbyists and academics back into the fold...

- Define the performance metrics for SMR.
  - Identify characteristic workloads
  - What can we quantify?
- Provide a starting point...
  - Reference Implementations
  - Documentation
  - Open Problems
  - Bounties and grants

Safe Memory Reclamation

Epoch Reclamation

- Amortizing protected section begin/end yields performance that is practically on-par with QSBR.
- Implemented lock-free and wait-free variants built on deferral scheme with thread-local retirement lists.
- Still vulnerable to unbounded memory growth.

Safe Memory Reclamation

PC, QSBR and EBR are all non-intrusive forms of SMR

PnP SMR

Read-Side

EBR	PC	QSBR

`epoch_read_begin();`
`[…]`
`epoch_read_end();`
	`pc_addref();`
	`[…]`
	`pc_delref();`
		`rcu_read_lock();`
		`[…]`
		`rcu_read_unlock();`

Write-Side (deferral)

EBR	PC	QSBR

`epoch_write_begin();`
`[…]`
`epoch_write_end();`
`epoch_defer(&a->e, destroy);`
	`[…]`
	`pc_defer(&a->e, destroy);`
		`[…]`
		`qsbr_defer(&a->e, destroy);`

- Do we want to generalize an interface?
- Can we create a general interface to these forms of SMR?

Summary of Opinions

- Documentation and reference implementations are underwhelming for neophytes (exception is RCU and QSBR).
- No metrics and standards of quality for prospective contributors.
- NBDS is vulnerable to SMR-scheme lock-down, difficult to plug and play schemes for various workloads.
- There is a lack of independent surveys.

Data Structures

appnexus

Data Structures

State of the World

Structure	SPSC	SPMC	SPNC	UPMC	MPMC	MPSC	MPNC
bag		○/○					
bitmap					○/○		
fifo	○/○			●/●			
ht		○/●					
stack				●/●	●/●		○/-
ring	○/○						
queue.h	○/○						

○ = WF ● = LF

Format is <producer>/<consumer>

Data Structures

Hash tables for unmanaged languages

- Bytestring keys allow for program-defined lifetime semantics
- Well-defined semantics
- Supports replace, insert, growth, delete, get, iteration
- Power-of-2 hash table with linear-probing/double-hashing hybrid
- Statistically a wait-free hash table

Future Work

- Provide an MPMC variant
- Trim down constant factors associated with interface
- Allow for user-defined upper-bounds on retry probability
- Allow bytestring tables to emulate memory savings of pointer-packing
- Expose memory hierarchy to probe sequence

Data Structures

Future Work

- RADIX tree
- Set
- Skiplist
- T-Tree (SPMC)

Summary of Opinions

- First instinct is to generalize to MPMC while interesting design patterns and avenues can be found through SPMC
- Allow applications to define lifetime of referenced objects with minimal data duplication
- General purpose libraries should allow for workload specialization
- Reference implementations needed for the classics
- Expose rank in wait-free hierarchy

The End

http://concurrencykit.org
http://appnexus.com

appnexus

Legal