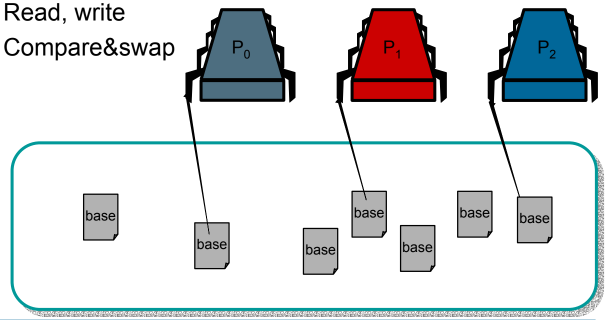
Time and Space Lower Bounds for Implementations Using *k*CAS

Hagit Attiya, Technion
Danny Hendler, University of Toronto

Shared-Memory Multi-processors

Asynchronous processes communicate through shared base objects, using:

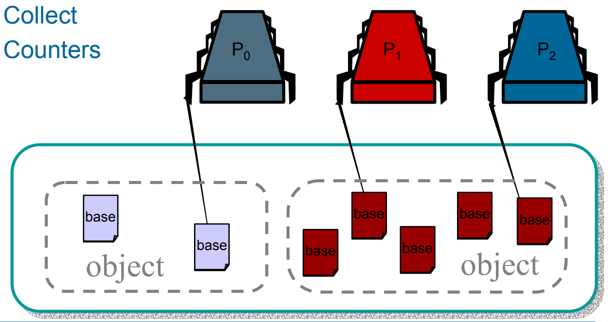
- Read, write
- Compare&swap



Implementing High-Level Objects

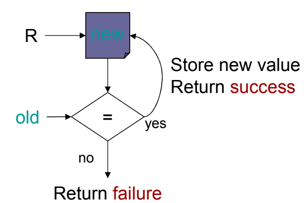
Base objects are encapsulated within other objects

- Stacks, queues
- Collect
- Counters



CAS Operations

- Atomically check and modify a base object




```
CAS(R, old, new)
atomically
  v ← read from R
  if (v = old) {
    R ← new
    return success
  }
  else return failure
```

CAS Operations

- Atomically check and modify a base object

Motorola 680x0
IBM 370
Sun SPARC
80X86

CAS(R, old, new)
atomically
  v ← read from R
  if (v = old) {
    R ← new
    return success
  }
  else return failure

kCAS Operations

- Atomically check and modify k base objects
- E.g, DCAS (k=2)

v_1 v_2
R_1 R_2

DCAS(R_1,R_2,old_1,old_2,new_1,new_1)
atomically
  v_1, v_2 ← read from R_1, R_2
  if (v_1 = old_1 and v_2 = old_2) {
    R_1, R_2 ← new_1, new_1
    return success
  }
  else return failure

Does Arity Matter?

- In software, CAS can implement any kCAS
  - E.g., software transactional memory [Shavit, Touitou]
  - Allows to solve the same problems, so computationally
    - CAS no stronger than DCAS
    - DCAS no stronger than 3CAS, etc.
- But at a cost…
  - Significant cost also for implementing kCAS in hardware

Is kCAS worth its cost?

Simplifies programming of practical data structures [Agesen et al.][Greenwald]

Some separation lower bounds [Attiya, Dagan]

Our Results: Step and Space Bounds

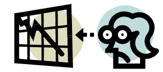
- Step complexity bounds
  - Reading kCAS reduces step complexity compared with CAS
  - Non-reading kCAS does not
- Space bounds
  - Reading kCAS does not reduce space complexity compared with CAS
  - Modifying kCAS increases space complexity compared with CAS

Counting w/ kCAS

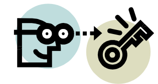
- Takes $\Omega(\log_k n)$ steps on the average
  - Holds even if kCAS returns the old values in these k locations (reading kCAS)
  - Extends an $\Omega(\log_2 n)$ worst case lower bound, for reads, writes and unary CAS (actually, LL/SC) [Jayanti]
- Lower bound is tight
  - An algorithm that collects information up a k-ary tree [Afek, Dauber, Touitou]
  - $O(\log_k n)$ worst-case step complexity

What About Non-Reading kCAS?

- A non-reading kCAS returns only a Boolean success/fail indication
  - Algorithm no longer works…



- A lower bound of $\Omega(\log_2 n)$ for collecting information
  - With reads, writes and non-reading kCAS
  - Regardless of k



Lower Bound for Non-Reading kCAS

Fan-in arguments do not apply
- Outcome depends on k objects

Use an information-theoretic argument
- A kCAS operation only gives a single bit of information
- Many bits are necessary to completely describe an input vector
- Still a fan-in argument on reads

Formalizing the amount of information obtained using process and object partitions of the input
- Introduced for CRCW PRAM [Beame]

Partitions of Input Vectors

Consider synchronized executions, so that in each round:
- A base object is modified by (at most) one process
- A process reads (at most) one base object modified in this round

PV(p,t): Possible states of process p after t steps
P(p,t): Partitioning of input vectors into equivalence classes
- If they lead p to the same state after t steps

At the end of its computation, a process is in a different state for every input vector (so it can return a different value)
- Each input vector is in a separate equivalence class
- There are 2^n input vectors (hence, classes)
- ⇨ The process must have 2^n possible termination states

Bounding the Growth of Partitions

C(R,t): Partitioning of the input vectors into equivalence classes, by the state of base object R after t steps

Size of P(p,*) and C(R,*) grows slowly with t

Case analysis, by type of base operation, e.g.

CAS → Multiplied by 2 (possible outcome)

read → Multiplied by C(R,t) (number of possible values)

⇨ Logarithmic (base 2) lower bound on the number of steps until number of possible states is 2^n

Does kCAS Reduce Space Complexity?

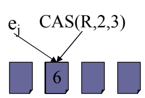
- Not really…
  - Even w/ reading kCAS
  - For a large class of problems
    - Collect
    - Counters
    - Stacks, queues, hash-tables
    - Swap
- Extends a space lower bound for the same class [Fich, Hendler, Shavit]

State w/ Levelled Sequence

Stripped-down definition
- only kCAS operations
- modify all base objects

Extends the notion of writes covering a set of objects [Burns, Lynch]

A sequence of kCAS events e_1, e_2, e_3, … by different processes,

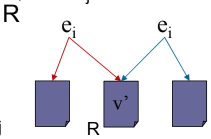


- Each event is visible by itself
  - Writes to some base object
- If i < j, then e_i is visible both in e_ie_j and in e_je_i
  - e_j does not over-write e_i (in e_ie_j)
  - e_j does not change the value of e_i's precondition (in e_je_i)

Locations Accessed in Levelled States

In a levelled state, kCAS events access disjoint base objects

For instance, assume two DCAS events e_i and e_j , i < j, are pending on the same base object R



- e_i is visible alone
- e_i is visible alone

⇨ v is the pre-condition for both e_i and e_j

e_j writes a value v' ≠ v
- e_i is not visible in e_ie_j

⇨ Contradiction

Space Bound with kCAS Only

Any collect object has an n–levelled state

[Fich, Hendler, Shavit]

⇨ Space complexity ≥ $k \times n$ if only kCAS is used

Can be extended to implementations mixing writes, CAS, DCAS, kCAS, …

- A smaller lower bound, though
- Indicates that it is best to use only CAS

Wrap-Up…

- The proven benefits of kCAS are limitted…
- We talked about CAS
  - Results hold for other conditional operations
- The benefit of modifying k locations atomically
  - *reading kCAS* vs. *non-reading kCAS & k-read*
- Other problems?
  - Lower bounds hold for a large class of objects (~perturbable)
  - What about one-shot problems, esp. consensus?