Algorithms that Adapt to Contention

Hagit Attiya

Department of Computer Science
Technion

Fast Mutex Algorithm

[Lamport, 1986]

- In a well-designed system, most of the time only a single process tries to get into the critical section…
- Will be able to do so in a constant number of steps.

When two processes try to get into the critical section? $O(n)$ steps!
Asynchronous Shared-Memory Systems

Need to collect information in order to coordinate…
When only few processes participate, reading one by one is prohibitive …

Talk Outline

- How to be adaptive in a global sense?
  - The splitter and its applications: renaming and collect.

- How to adapt dynamically?
  - The sieve and its applications: renaming and collect.

- Extensions and connections.
Adaptive Step Complexity

- The step complexity of the algorithm depends only on the number of active processes.

**Total** contention: The number of processes that (ever) take a step during the execution.

---

**A Splitter**

[Moir & Anderson, 1995]

A process stops if it is alone in the splitter.
Splitter Implementation


1. \( X = \text{id}_i \) // write your identifier
2. if \( Y \) then return( right )
3. \( Y = \text{true} \)
4. if \( ( X == \text{id}_i ) \) // check identifier
   then return( stop )
5. else return( left )

Requires \( O(1) \) read / write operations, and two shared registers.

Things to do with a Splitter

- A triangular matrix of splitters.
- Traverse array, starting at the top left, according to the values returned by splitters
- Until stopping in some splitter.
Things to do with a Splitter

- One process does not follow each direction.
- After \( \leq k \) movements, a process is alone in a splitter.
- A process stops at row, column \( \leq k \)
- At most \( O(k) \) steps.

Things to do with a Splitter: \( k^2 \)-Renaming

Diagonal association of names with splitters.

\( \Rightarrow \) Take a name \( \leq k^2 \).
Even Better Things with a Splitter: Store

- Associate a register with each splitter.
- A process writes its value in the splitter where it stops.
- *Mark* a splitter if accessed by some process.

Even Better Things with a Splitter: Collect

- Associate a register with each splitter.
- The current values can be collected from the associated registers.
- Going in diagonals, until reaching an unmarked diagonal.
Even Better Things with a Splitter: Store and Collect

- The first store accesses $\leq k$ splitters.
- A collect may need to access $k^2$ splitters...

Improve?

Binary Collect Tree
Binary Collect Tree

- To store:
  traverse the tree until stops in some splitter.
  Later, write in the register associated with this splitter.

- To collect:
  DFS traverse the marked tree, and read the associated registers.
  Marked tree contains $\leq 2^{k-1}$ splitters.
Size of Marked Sub-Tree

In a DFS ordering of the marked sub-tree,
There is $\geq$ one acquired node (where a process stops),
Between every pair of marked nodes.

Simple Things to Do with a Linear Collect

- Every algorithm with $f(k)$ iterations of collect and store operations can be made adaptive.
  - Atomic snapshots
    - $O(k)$ iterations.
    - $\Rightarrow O(k^2)$ steps.
  - Renaming.
  - ...

[Afek et al. 1991]
More Sophisticated Things to Do with a Linear Collect

- At each spine node:
  - Collect.
  - If # processes ≤ label
    - continue left
  - Else
    - continue right
    - remember values.

More Sophisticated Things to Do with a Linear Collect

- At most 2, 4, 8, etc. processes move to the left sub tree.
- # participants in a sub-tree is bounded.
- Perform an ordinary algorithm in sub-tree.
More Sophisticated Things to Do with a Linear Collect

- If move right, at least 2, 4, 8, \ldots participants.

\[ \Rightarrow \text{The step complexity is justified.} \]

More Sophisticated Things to Do Efficient Atomic Snapshot

- E.g., atomic snapshot algorithm.
  \cite{Attiya & Rachman, 1998}

\[ \Rightarrow \text{An } O(k \log k) \text{ atomic snapshot algorithm.} \]
Be More Adaptive?

- In a *long-lived* algorithm…
  …processes come and go.

- What if many processes start the execution, then stop participating?
  …then starts again…
  …then stops again…

Who’s Active Now?

Interval contention during an operation: The number of processes (*ever*) taking a step during the operation.

[Afek, Stupp & Touitou, 1999]
Who’s Active Now?

Point contention of an operation: Max number of processes taking steps together during the operation.

Clearly, point contention $\leq$ interval contention.

Catching Processes with a Sieve

- A dynamic object, built for repeated usage.
- When a set of processes access the sieve concurrently at least one is caught by the sieve.
- Good synchronization properties.
Sieve: More Formally

- Returns a view of processes accessing the sieve concurrently.
  - A non-empty set of candidates $C$.
  - A process returns either $\phi$ or $C$.
  - At least one process $p_w \in C$ returns $C$. $p_w$ is a winner.

Sieve Properties

- Agreement on the set of candidates (safety).
  - Synchronization.
  - Exchange of information.

- Can be filled and emptied many times (long-lived).

- An empty sieve catches at least one process (liveness).
  - Adapting to point contention.
Sieve Implementation: Copies

- Count

1 2 3 4 5 6

Current copy of the sieve

Sieve Implementation: Doorway

- Restrict access to a copy only to simultaneously active processes.
- # processes inside the copy ≤ point contention.
- Inside the copy, employ algorithms adaptive to total contention!
Sieve Implementation: Candidates

take an atomic snapshot
write the returned view
find minimal view C
if all processes in C wrote their view
return C
else return $\phi$

[Borowsky & Gafni]

Sieve Implementation: Winners

take an atomic snapshot
write the returned view
find minimal view C
if all processes in C wrote their view
return C
else return $\phi$
Sieve Implementation: Winners

take an atomic snapshot
write the returned view
find minimal view $C$
if all processes in $C$
  wrote their view
return $C$
else return $\phi$

Last process in $C$ to write a
view is a winner.

Sieve Implementation: Managing the copies

- Candidates are synchronized (work together).
  - Increase $\text{count}$ by 1.
  - Monotone...

- When all candidates leave a copy, open the next copy.
Things to do with a Sieve: $2k^2$–Renaming

Place sieves in a row…

1  2  3  4  2n – 1

return (4, rank in C)

Agreement on set of candidates and uniqueness of copies.

$\Rightarrow$ **Uniqueness** of names.

---

Things to do with a Sieve: $2k^2$–Renaming

Potential method shows that a process skips $\leq 2k-2$ sieves.

- $k$ is the point contention.

Wins in sieve $\leq 2k-1$. 
Things to do with a Sieve:

**2k^2**–Renaming

- Wins in sieve $\leq 2k-1$.
- $O(f(k))$ step complexity.
- C includes at most $k$ processes.
- Name $\leq 2k^2$.

---

Things to do with a Sieve:

**Store**

Place sieves in a row…

- Agreement on set of candidates and uniqueness of copies
  - $p_w$ writes the values of all candidates in a register associated with the sieve.
Things to do with a Sieve:

Collect

- Go over the associated registers and read…

Adaptive Collect?

- $p_w$ and all other operations complete.
- A collect still has to reach the splitter in which $p_w$ has written its value!
Bubble-Up

[Afek, Stupp & Touitou, 2000]

- Before completing an operation, move information from far away sieves to the first sieve.

Other Things to do with a Sieve

- Sieve-based collect can be used to implement:
  - Atomic snapshots.
  - Immediate snapshots.
- Timestamps: The vector of copy numbers.
  - Long-lived renaming.
    - Polynomial step complexity (using collect).
What About Mutex?

- Cannot have adaptive step complexity…
- Can have adaptive system response time.
  
  [Attiya & Bortnikov, 2002]
  
  - Some techniques are similar.
  - Renaming, adaptive binary tree (bottom-up)…

What’s Next?

- Other problems.
  - Snapshots, generic simulations…
- Improve and simplify.
  - Find good building blocks.
  - Local step complexity!
- Use stronger primitives (C&S, LL/SC).
  
  [Afek, Dauber & Touitou, 1995]
Uniform Algorithms

[Gafni, 2001]

- Adaptive algorithms can be also considered as algorithms that do not depend on the number of participants.
- Useful in the context of peer-to-peer systems, with no centralized control.
  - A huge number of potential processes.
  - Join and leave…

Space: The New Frontier

- Our results are based on a collect algorithm.
  - Either $O(K^2)$ step complexity ($K$ is total contention),
  - Or exponential space complexity.
- A better collect algorithm?
  - $O(K)$ step complexity, and
  - Polynomial space complexity.
- A lower bound proof?
The Whole Story

This talk describes some of the results in:
