Anonymous Processors with Synchronous Shared Memory: Monte Carlo Algorithms

Bogdan Chlebus[1] Gianluca De Marco [2] Muhammed Talo [3]

[1]University of Colorado Denver, USA

[2]Università degli Studi di Salerno, Italy

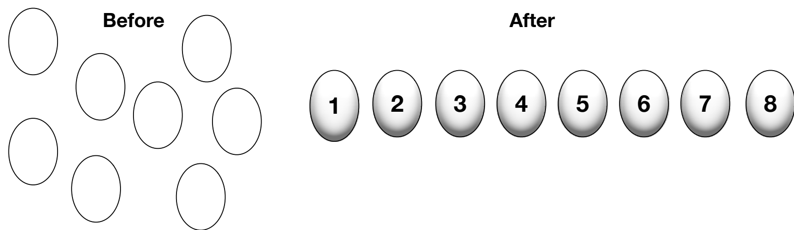
[3]University of Munzur, Turkey

OPODIS 2017
Lisboa, Portugal

The Name of the Game

- **Processes need unique identifiers:** to execute deterministic algorithms.
- We consider distributed systems in which individual processors are **anonymous** and completely indistinguishable.
- ***Ultimate goal:*** to assign individual names by all the processors to themselves.
- The processors execute a **randomized distributed algorithm** to assign names to themselves

Anonymous processes/nodes name themselves



This job accomplished by the processors/nodes themselves.

No external help?

There is a source of random bits.

Randomized algorithms

1. A randomized algorithm is ***Las Vegas*** when it terminates almost surely and the algorithm returns a **correct output** upon termination.

2. A randomized algorithm is ***Monte Carlo*** when it terminates almost surely and an **incorrect output may be produced** upon termination, but the probability of error converges to zero with the size of input growing unbounded.

Parallel Random Access Machine (PRAM)

A distributed system in which

- n processors communicate using read-write shared memory.
- operations performed on shared memory occur synchronously

  (executions of algorithms are structured as sequences of globally synchronized rounds)
- Each processor is an independent random access machine.

Common PRAM

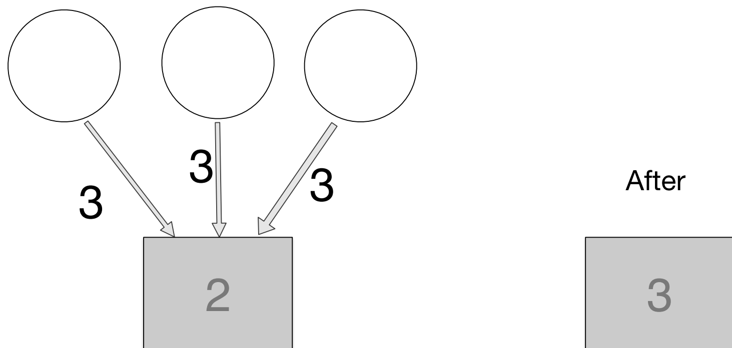


Figure: Common PRAM: only equal values may be concurrently attempted to be written into a register.

Arbitrary PRAM

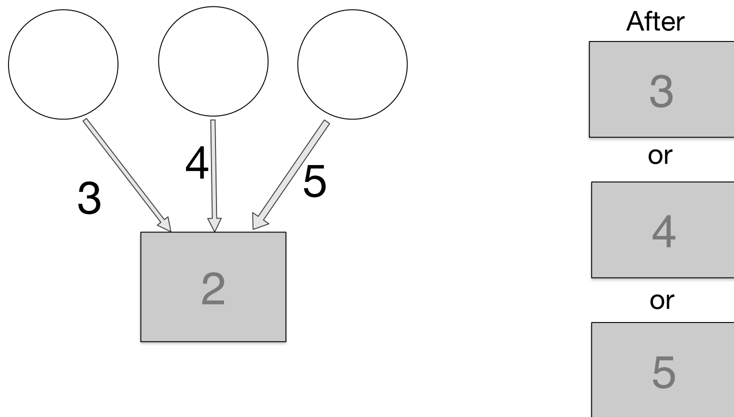


Figure: Arbitrary PRAM: distinct values may be concurrently attempted to be written into a register, one of them (selected arbitrarily) gets written.

Example: pseudocode for Common PRAM

Procedure: VERIFY-COLLISION (x)

```
initialize    Heads[x] ← Tails[x] ← false
toss ← outcome of tossing a fair coin
if toss = tails then Tails[x] ← true
      else Heads[x] ← true
return Tails[x] = Heads[x]
```

Address x determined by a processor's random choice.

`Heads` and `Tails` are arrays of shared memory cells.

The procedure returns `true` when a collision has been detected: more than one processors play with this x.

Four naming problems for PRAM

Two independent parameters determine a naming problem:

- amount of shared memory
  1. a **constant** number of memory cells
  2. **unbounded** shared memory
- PRAM variant
  1. **Arbitrary** PRAM
  2. **Common** PRAM

How randomness used in algorithms

Balls into bins

- Choosing tentative names can be visualized as throwing balls into bins.
- writing into register x: places a ball into x
  (works for unbounded memory)
- writing to fixed register in round x: places a ball in x
  (works for bounded memory)

Lower bound on random bits and probability of error

Proposition

If a randomized naming PRAM algorithm executed by n anonymous processors is correct with some probability p_n then it requires $\Omega(n \log n)$ random bits with the same probability p_n.

Proposition

For unknown n, if a randomized naming algorithm is executed by n anonymous processors, then an execution is incorrect, in that duplicate names are assigned to distinct processors, with probability that is at least $n^{-\Omega(1)}$, assuming that the algorithm uses $\mathcal{O}(n \log n)$ random bits with probability $1 - n^{-\Omega(1)}$.

Lower bound on time for Arbitrary & Common PRAM

Theorem

A randomized naming algorithm for an Arbitrary PRAM with n processors and $C > 0$ shared memory cells operates in $\Omega(n/C)$ expected time when it is either a Las Vegas algorithm or a Monte Carlo algorithm with the probability of error smaller than $1/2$.

Theorem

A randomized naming algorithm for a Common PRAM with n processors and $C > 0$ shared memory cells operates in $\Omega(n \log n/C)$ expected time when it is either a Las Vegas algorithm or a Monte Carlo algorithm with the probability of error smaller than $1/2$.

This fact is useful for amount of memory $C = \mathcal{O}(1)$

Logarithmic lower bound on time

Theorem

A randomized naming algorithm for a PRAM with n processors operates in $\Omega(\log n)$ expected time when it is either a Las Vegas algorithm or a Monte Carlo algorithm with the probability of error smaller than $1/2$.

Holds for both Common and Arbitrary PRAM variants.

This fact is useful for unbounded memory.

Algorithm: Monte Carlo for Arbitrary Bounded

Auxiliary β-*process* of throwing **balls into bins**,

- The process proceeds through stages identified by consecutive positive integers.
- During a stage, we first throw n balls into the corresponding 2^k bins and next count the number of occupied bins.
- β-process terminates, when the number of occupied bins is smaller than or equal to $2^{k/\beta}$.
- β-process is *correct* when upon termination each ball is in a separate bin, otherwise the process is *incorrect*.

Pseudocode: Arbitrary-Constant-Monte-Carlo

```
initialize  k ← 1
repeat
    initialize  Counter ← name_v ← 0
    k ← 2k
    x ← random integer in [1, 2^k]
    repeat
        All-Named ← true
        if  name_v ≤ 0 then
            Pad ← x
            if  x = Pad then
                Counter ← Counter + 1  ;  name ← Counter
            else All-Named ← false
    until All-Named
until  Counter ≤ 2^{k/β}
```

Performance: Arbitrary-Constant-Monte-Carlo

Theorem

The algorithm always terminates, for any $\beta > 0$. For each $a > 0$ there exists $\beta > 0$ and $c > 0$ such that the algorithm assigns unique names, works in time at most cn, and uses at most $cn \ln n$ random bits, all this with probability at least $1 - n^{-a}$.

Optimality with respect to the following performance measures:

- the expected time $\mathcal{O}(n)$.
- the expected number of random bits $\mathcal{O}(n \log n)$.
- the probability of error $n^{-\mathcal{O}(1)}$.

Monte Carlo Algorithms

PRAM Model	Memory	Time	Algorithm
Arbitrary	$\mathcal{O}(1)$	$\mathcal{O}(n)$	Arbitrary-Bounded-MC
Arbitrary	unbounded	polylog	Arbitrary-Unbounded-MC
Common	$\mathcal{O}(1)$	$\mathcal{O}(n \log n)$	Common-Bounded-MC
Common	unbounded	polylog	Common-Unbounded-MC

Table: When time is marked as "polylog" this means that the algorithm comes in two variants, such that in one the expected time is $\mathcal{O}(\log n)$ and the amount of used shared memory is suboptimal $n^{\mathcal{O}(1)}$, and in the other the expected time is suboptimal $\mathcal{O}(\log^2 n)$ but the amount of used shared memory misses optimality only by at most a logarithmic factor.

Open Problems and Future Work

It is an open problem to develop **Monte Carlo** algorithms for Arbitrary and Common PRAMs for the case when the amount of shared memory is unbounded, such that they are **simultaneously asymptotically optimal** with respect to

- expected time,
- expected number of generated random bits
- probability of error.

This is all. **Thank you!**