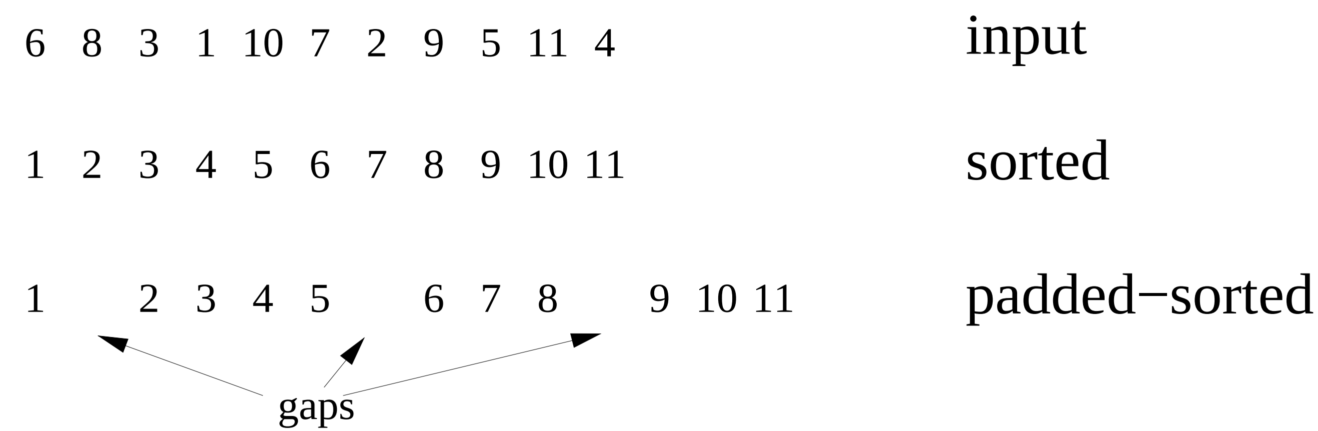
Approximate Compaction and Padded-Sorting on Exclusive Write PRAMs

Mirosław Kutyłowski (Paderborn), Tomasz Wierzbicki (Wrocław)

IPPS'96

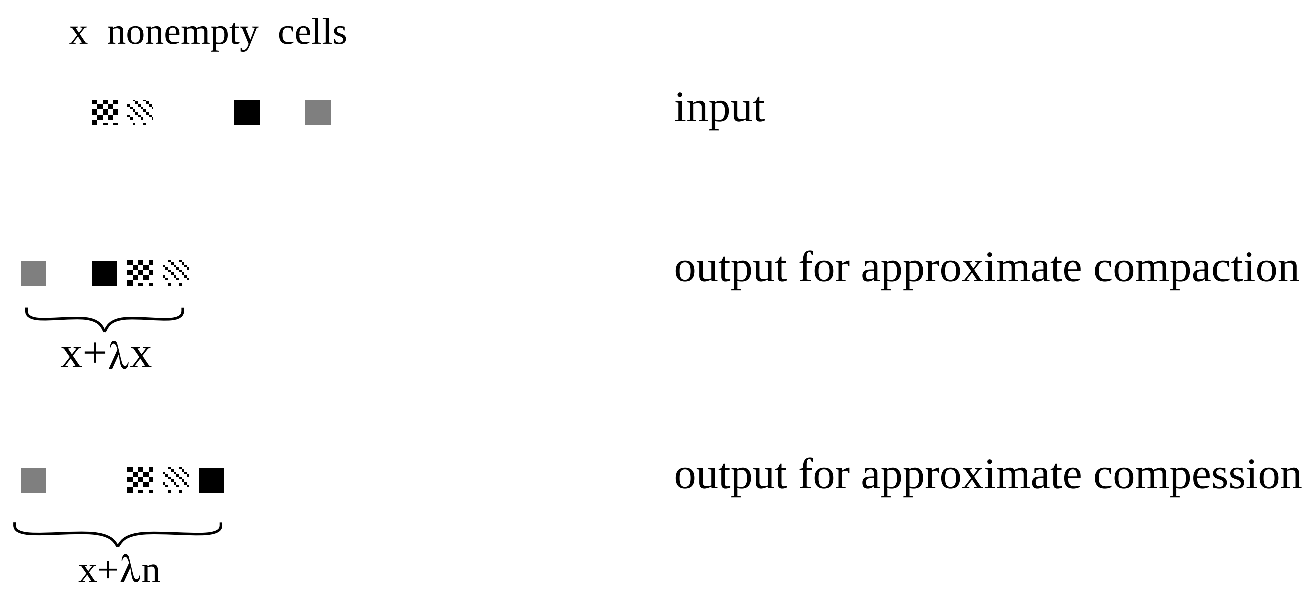
Motivation:

- exact sorting not always necessary
- small number of gaps can be tolerated



Related problems

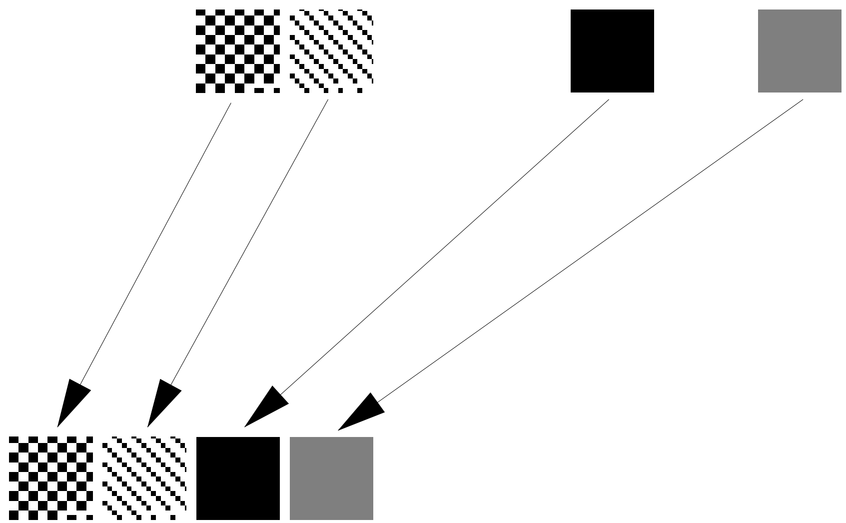
- approximate compaction
- approximate compression



Sequential case

Padded-sorting (approximate compaction, approximate compression) does not make sense.

In $O(n)$ steps the gaps might be removed.



Parallel sorting is difficult

the problem of sorting is difficult, even on CRCW PRAM with $n^{O(1)}$ processors the runtime is

$$\Omega(\frac{\log n}{\log\log n})$$

(Beame, Håstad)

Parallel padded-sorting is easier

- **runtime** $O(\log^* n)$ (Hagerup,Raman)
  n-processor randomized PRIORITY CRCW PRAM,
  the input elements chosen independently and uniformly from interval $[0, 1]$,
  λ- constant
- **runtime** $\Omega(\log^* n)$ (MacKenzie)
  even for constant λ,
  on n-processor randomized PRIORITY CRCW PRAM
- **using comparisons only: runtime** $O(\log n / \log k)$ (Hagerup,Raman)
- **deterministically: runtime** $(\log n / \log k) \cdot (\log \log k)^3 \cdot 2^{O(\log^* n - \log^* k)}$
  (Goldberg, Zwick)
  with kn processors on a COMMON CRCW PRAM.

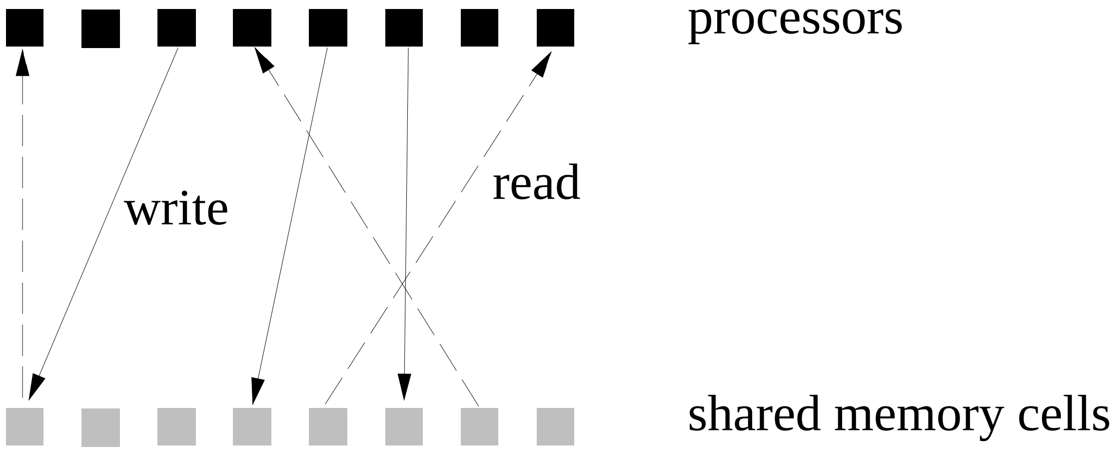
Problems with PRAMs

- routing the messages in a constant time
  regarded as unrealistic
- congestion
  read or write requests to one location require some time to be serviced

Problem

Do we have fast padded-sorting algorithms if we assume that **no congestion** occurs?

EREW model

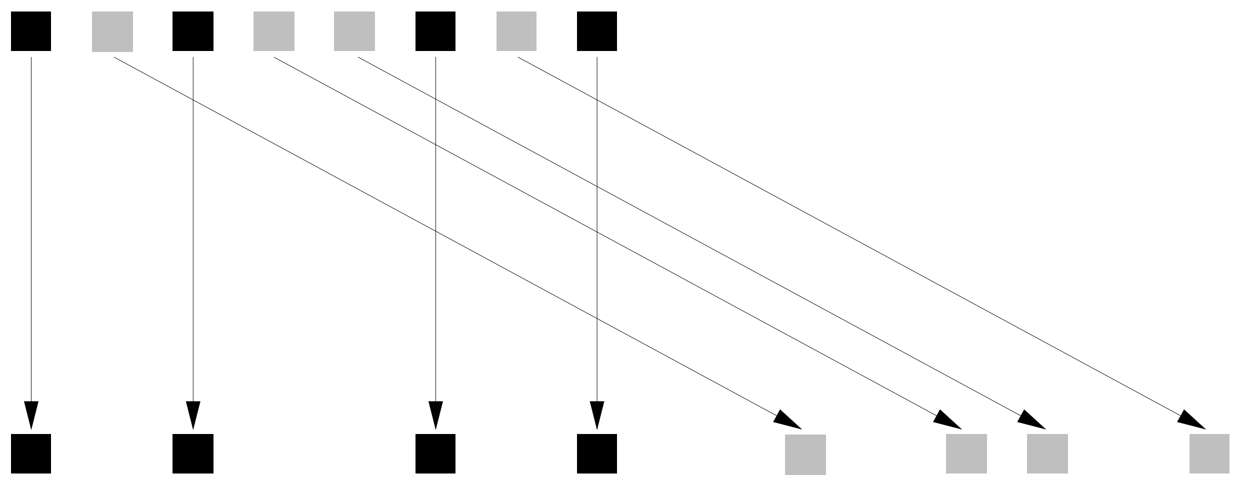


A parallel step:

- read phase – each processors may read an arbitrary location, but no read conflict may occur,
- internal computations phase,
- write phase – each processors may write into arbitrary locations, but no write conflict may occur.

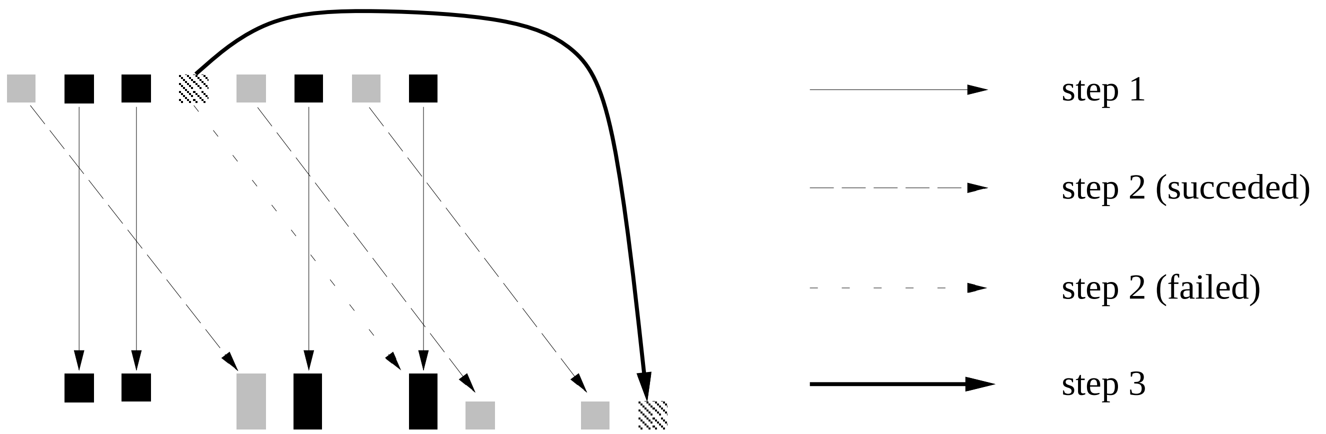
BUCKETSORT

- input: string of length n consisting of 0's and 1's
- output: padded-sorted string of length $2n$
- time: $O(1)$



Bucketsort and less gaps

- padded-factor smaller
- runtime bigger



This method is optimal

Let $T(n)$ = runtime for n-bit strings

Results (simplified for binary inputs):

- for $\lambda < 1$:

$$T(n) \geq \frac{1}{4} \log((1-\lambda)n) - O(1)$$

- for $\lambda \leq \frac{1-\varepsilon}{2}$ ($\varepsilon > 0$ is an arbitrary constant):

$$T(n) \geq 0.71 \log n - O(1)$$

Example

To achieve runtime $O(\log\log n)$ we have to take

$$\lambda \geq 1 - \frac{\log n}{n} \approx 1$$

So better use Bucketsort!

- The lower bounds hold for CREW PRAM as well (concurrent reads possible)
- The bounds can be generalized for input string of k elements.

Strings of arbitrary elements

Corollary.
For padded-sorting n rationals from the interval $[0,1]$ on CREW PRAM

$$0.71 \log n - O(1)$$

steps are required for any padding-factor λ.
(the same time as for sorting!)

Last chance – randomization

But there are **bad news**:

- almost no randomized algorithms known for EREW PRAM
- algorithms for EREW PRAM running in sublogarithmic time extremely rare
- complexities of a Boolean function on CREW PRAM and randomized CREW PRAM are the same (up to a constant factor)

Padded-sorting on randomized EREW

Upper bound:

Runtime $T(n)$ for padding factor λ satisfies

$$T(n) \leq 2\log(\lambda^{-1}) + O(\log\log n)^2$$

In particular, for $\lambda = n^{-\alpha}$

$$T(n) \leq 2\alpha \log n + O(\log\log n)^2.$$

Lower bound

A matching lower bound.

In particular, for $\lambda(n) = n^{-\alpha}$, $\alpha < 1$,

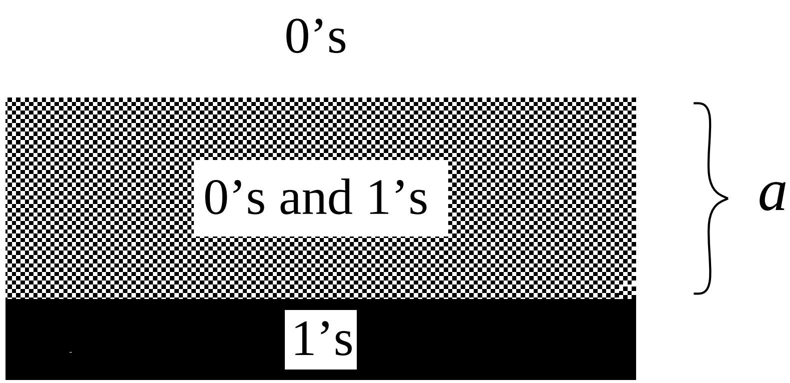
$$T(n) \geq \alpha \cdot 0.71 \log n - O(1).$$

The idea of the randomized algorithm

- arrange the elements in a 2-dimensional array with columns of height $\approx \lambda(n) \cdot n$
- perform several following rounds:
  - permute every row at random
  - sort (adaptively) the rows
- copy the ones from a region on the boundary between 0's and 1's elsewhere

Effects of a round

Initial situation



After the round: