PRAM algorithmics

Some problems and their complexity

Plan

1. Some indicators to evaluate a PRAM algorithm
–not studied here, see a future course
2. **Complexity of parallel problems**

The NC complexity class of parallel problems

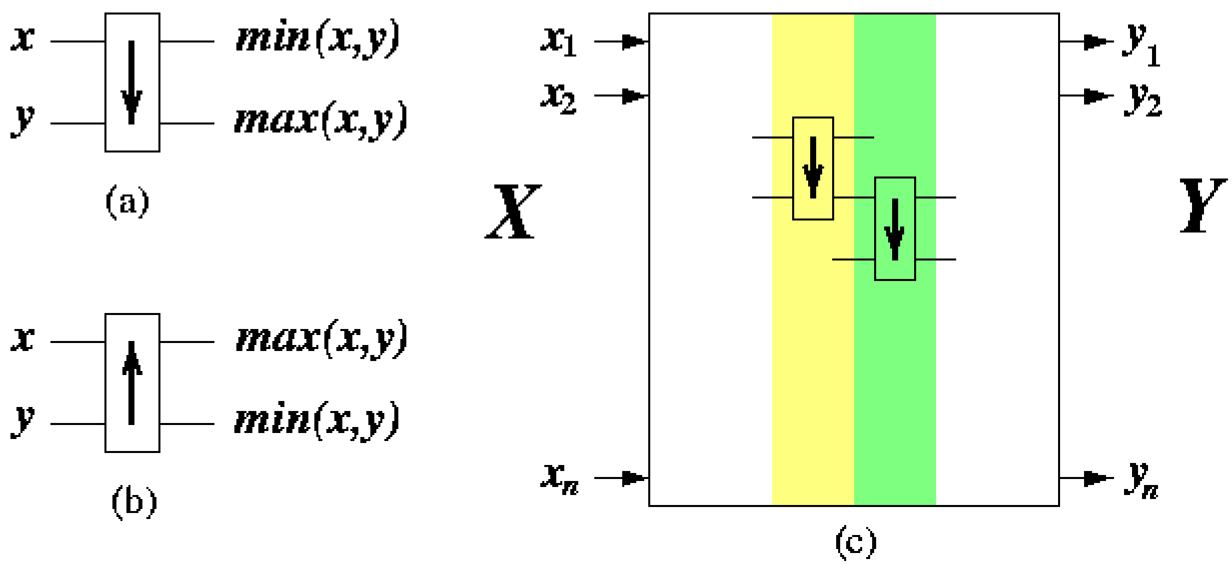
- It tell us what is a « good » parallel algo
- NC complexity class (« Nick's Class »)
  - The set of problems for which there exists a parallel algorithm taking a (poly)logarithmic parallel time, and using a polynomial number of processors
  - $\mathcal{NC}$ in $\mathcal{P}$, $\mathcal{P}$? in $\mathcal{NC}$, probably ≠
- An « Optimal » parallel algorithm :
  - Belongs to NC, and moreover is efficient (in work)
- Be careful in practice with the poly-log time:
  - Ex: A //time = $\log^3 n << n^{1/4}$ only when $n > 10^{12}$
- More: Parallel complexity theory - NC algorithms (wisc.edu) and Parallel complexity theory - P-completeness (wisc.edu)

Sort n values in an optimal way ?

- Complexity to sort n values:
  - Somehow, they all must be compared 2 by 2 (cf max-v1)
  - It is known that the lower bound in sequential is $\Omega(n*\log n)$
  - =>It provides us with a framework for parallel algorithms!
    - **With only O(n) procs. used, goal is to sort in O(logn) //time**
      - It is feasible, but the factor hidden in the O( ) is very high
      - <u>Principle of the merge parallel sort algo on an EREW [due to Cole] :</u>
        - Start from n lists of size 1, merge them two by two in //
        - Start again, to merge all these lists 2 by 2, and so on
        Depth of the tree to traverse from leaves to root: log n, so, the sub goal is to **merge two lists of length O(n) in constant time ….** It is hard but feasible ; make inactive procs of the upper stages in the tree become active in order to contribute to these merge operations in O(1) // time that run at lower stages: pipeline, anticipate
- In pratice: Sort in // in time $O(\log^2 n)$, non optimal
  - On a PRAM
  - Or, on a « sorting network » : it is a topology of *sorting elements* that is always the same whatever be the initial sequence of input data to be sorted
    - Provides an « oblivious » or « regular » algorithm (i.e., just dependant of the problem size, not of the data values)

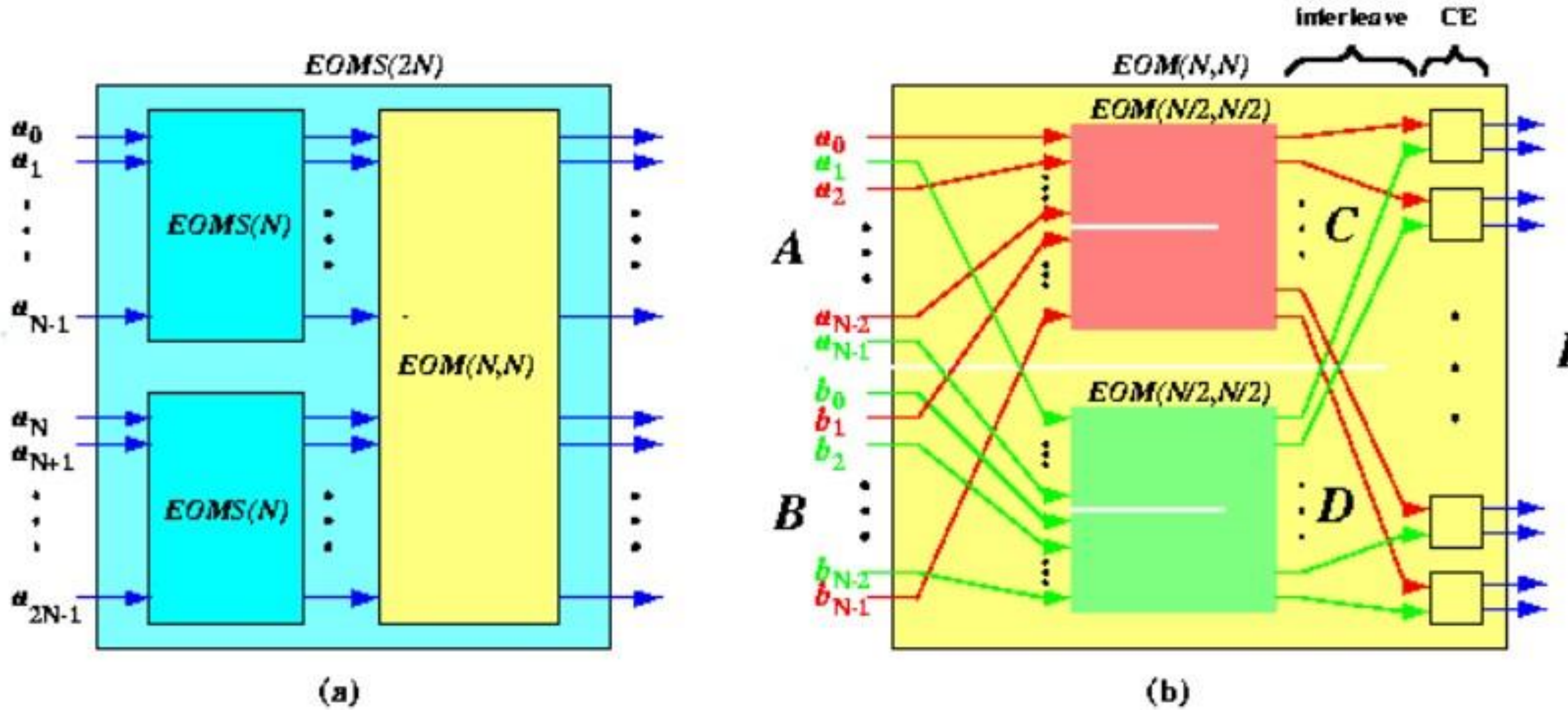
Architecture of a sorting network

- Built using 2x2 comparators/sorting elements
- Architecture of comparison-exchange sorting networks.
  - (a) The default type of comparator
  - (b) The second type of comparator.
  - (c) Sorting network composed from columns of basic comparators.



[Introduction to parallel sorting on mesh-based topologies (wisc.edu)]

Even-Odd Merge Sorting network



Even-Odd MergeSort (a) and Merge (b) network

C={EOMerge}(even(A),odd(B))

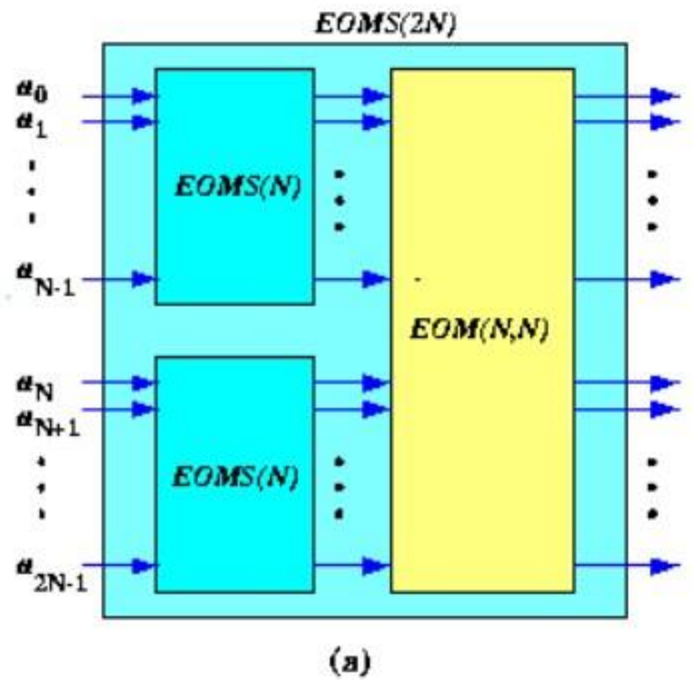
D={EOMerge}(odd(A),even(B))

L'={Interleave}(C,D)

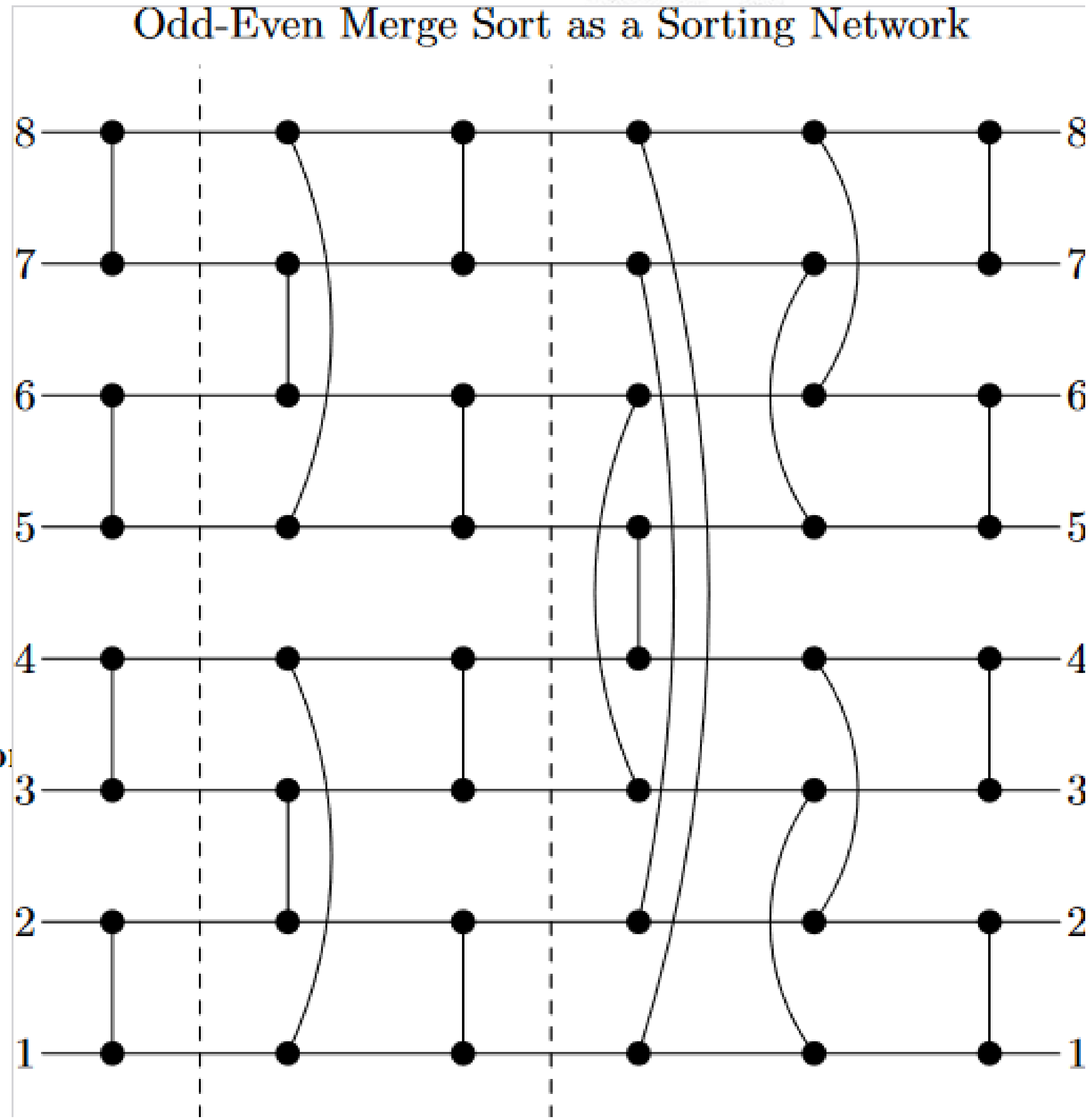
L={EOMerge}(A,B)={Pairwise_CE}(L')

[Sorting on hypercubic networks (wisc.edu)]

Complexity:
$\log^2 2N$ // time, $N.\log^2 2N$ comparators



Even-Odd MergeSort (a) and Merge (b) netwo



Log(2N) stages (dotted lines separated)
In each stage, for input of length k+k, log 2k stages to merge