Nonblocking On-Chip Interconnection Networks

Tripti Jain

Embedded Systems Chair
Department of Computer Science
TU Kaiserslautern, Germany

July 26, 2019

Table of Contents

1. Motivation
2. Known Solutions
3. Contributions
4. Experimental Results
5. Summary

Motivation

- **Moore's law:**

  number of transistors per mm^2 is doubling every year

↪ resulted in exponential growth of computational power

- around 2005, it has however reached its technological limits

↪ parallel architectures required for further performance increase

↪ parallel architectures are realized as systems-on-a-chip (SoC)

↪ new challenge:

developing efficient interconnection networks

Commonly Used Interconnection Networks

	architecture
AMBA 2.0,3.0 (ARM)	bus
CoreConnect (IBM)	bus
STBus (STmicro.)	bus
Eclipse (academic)	2D-mesh
Cliche	2D-mesh

- buses have limited bandwidth

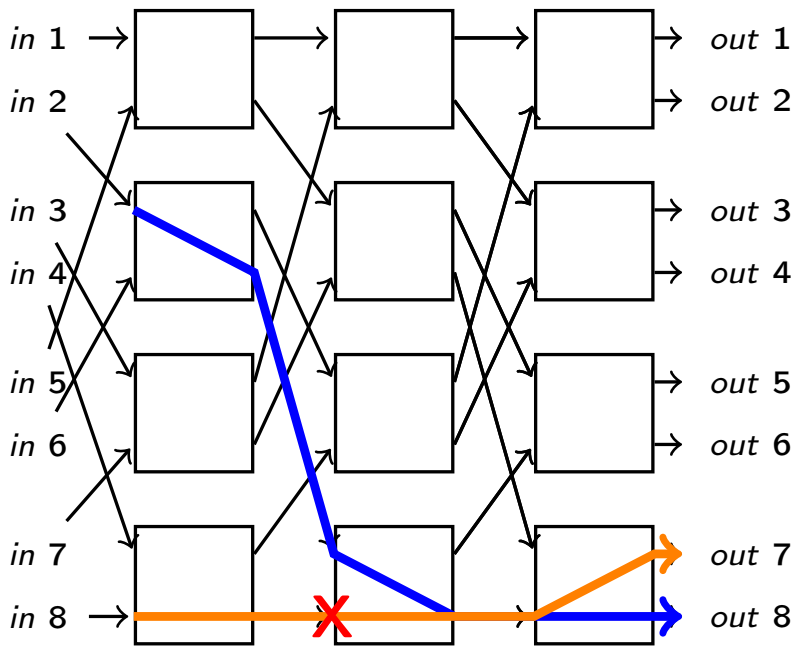
↪ buses are currently replaced with 2D-mesh networks

- but the latter are also limited, i.e., blocking (see next slides)
  - ↪ poor real-time guarantees
  - ↪ deadlocks have to be avoided by special routing algorithms

↪ **nonblocking networks are required for high-performance**

Blocking Networks – Example: Omega-Network

blocking networks cannot implement all $n!$ unicast connections

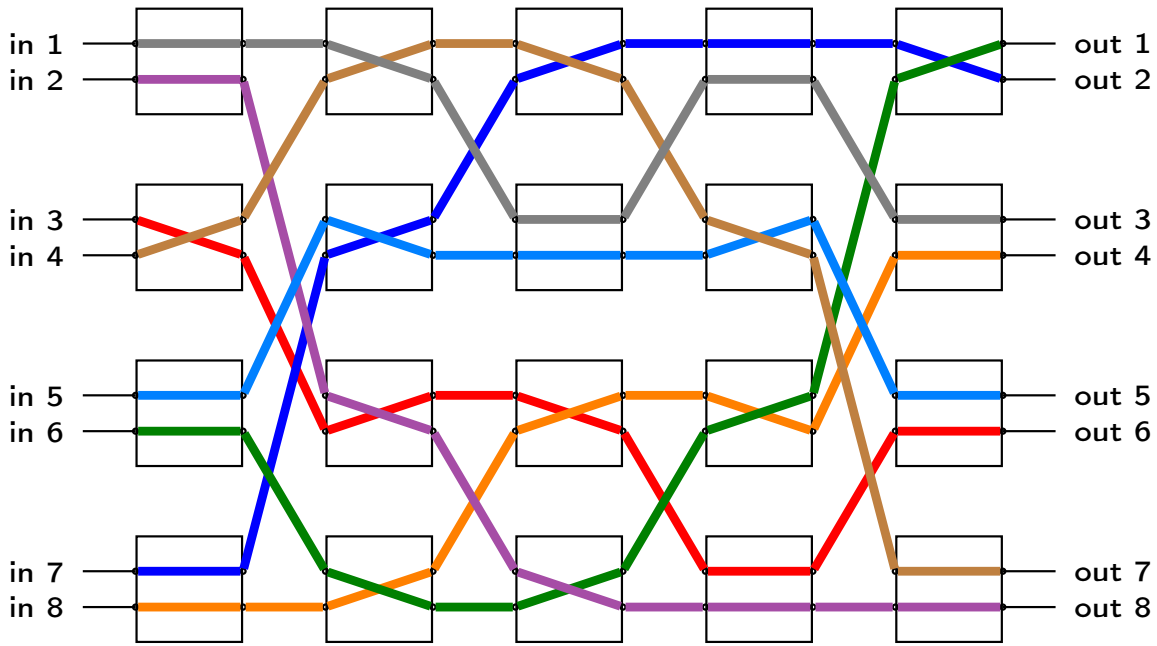


conflicting connections

- in 2 → out 8
- in 8 → out 7

Nonblocking Networks – Example: Beneš Network

nonblocking networks can implement all $n!$ unicast connections



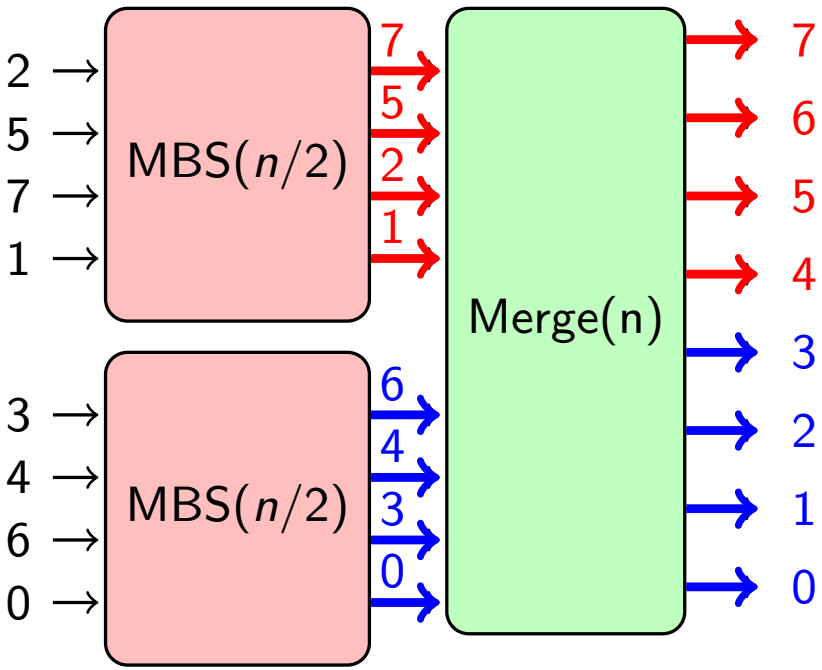
connections

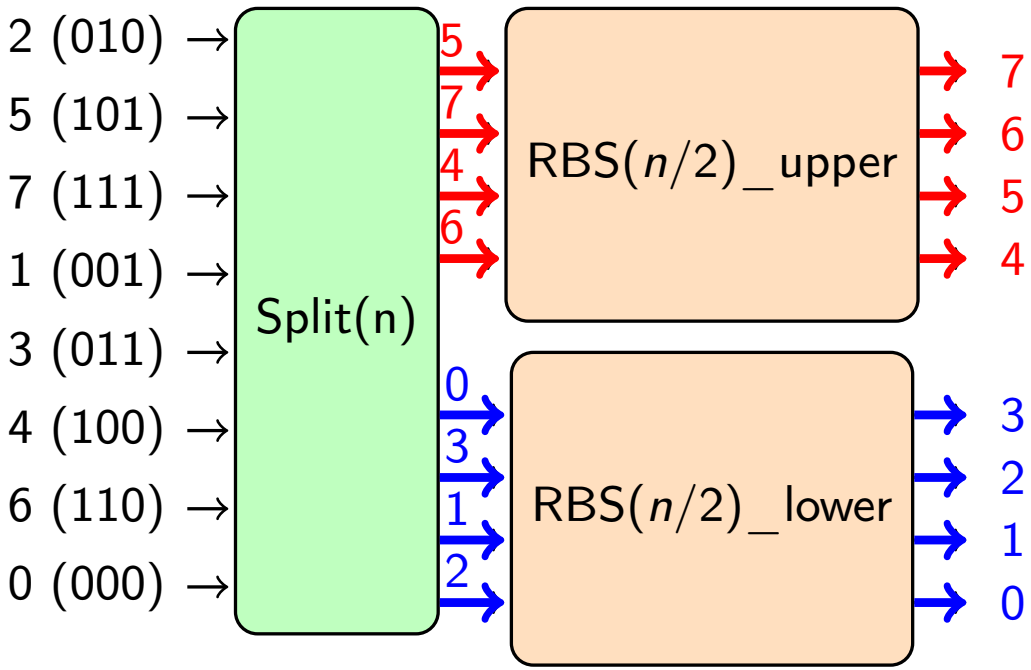
- in 3 → out 6
- in 7 → out 2
- in 8 → out 4
- in 6 → out 1
- in 2 → out 8
- in 5 → out 5
- in 4 → out 7
- in 1 → out 3

Known Nonblocking Networks

- **crossbars:**
  - prohibitive size of $O(n^2)$
  - simple routing algorithm
- **Beneš networks:**
  - scalable size of $O(n \log(n)^a)$
  - difficult routing algorithm
- **sorting networks:** (see next slide)
  - scalable size of $O(n \log(n)^a)$
  - routing := sorting by target addresses
  - but cannot route inactive inputs (will be discussed later)

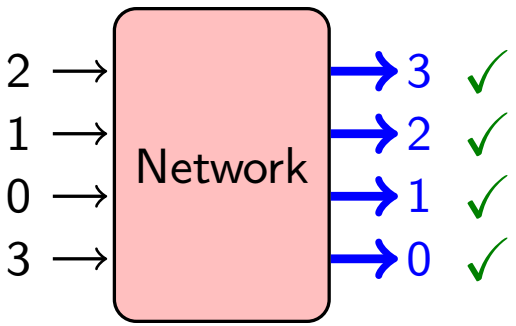
Sorting Networks as Interconnection Networks





- sort both halves of the given input sequence separately
- then merge the sorted halves
- distribute inputs according to their MSB in the right halves
- then halves are recursively sorted

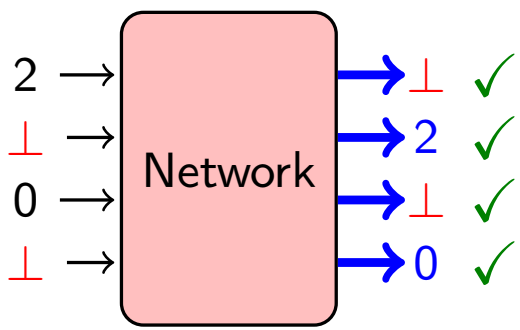
Problem for Sorting Networks: Partial Permutations



total permutations:

- every input is connected to a unique output

↬ routing = sorting by target addresses



partial permutations:

- some inputs have no connection (invalid inputs)

↬ inputs with invalid target address $\perp$

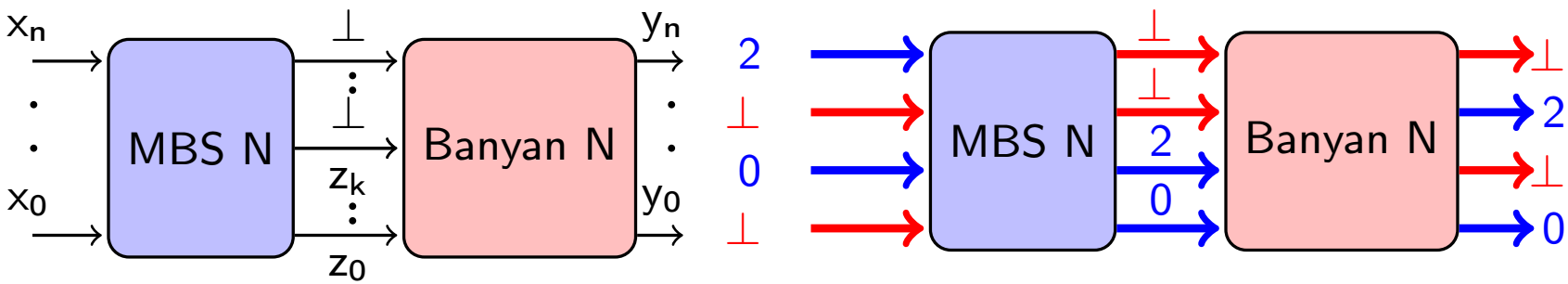
- in the example, we must have $\perp < 2 < \perp$

↬ no place for $\perp$ in the order is the right one

↬ **routing $\neq$ sorting target addresses**

Known Solutions (1): Batcher-Banyan Network

- Batcher-Banyan network[1]:
  add a Banyan network to the sorting network

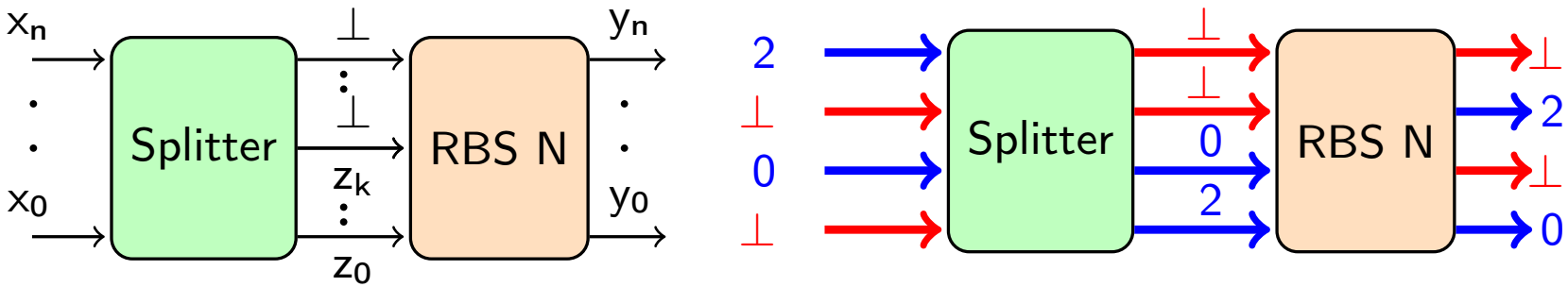


- for sorting, consider $\perp$ as the largest address
- additional Banyan network can route sequences $\alpha\perp^k$
- Batcher-Banyan works for all MBS networks,
  but unfortunately not for RBS sorting networks

[1] M. Narasimha. The Batcher-Banyan self-routing network: universality and simplification. *IEEE Transcations on Communications, 36(10):1175-1178, October 1988.*

Known Solutions (2): Narasimha's Network

- Narasimha[2] proposed a front-end splitter for a RBS network



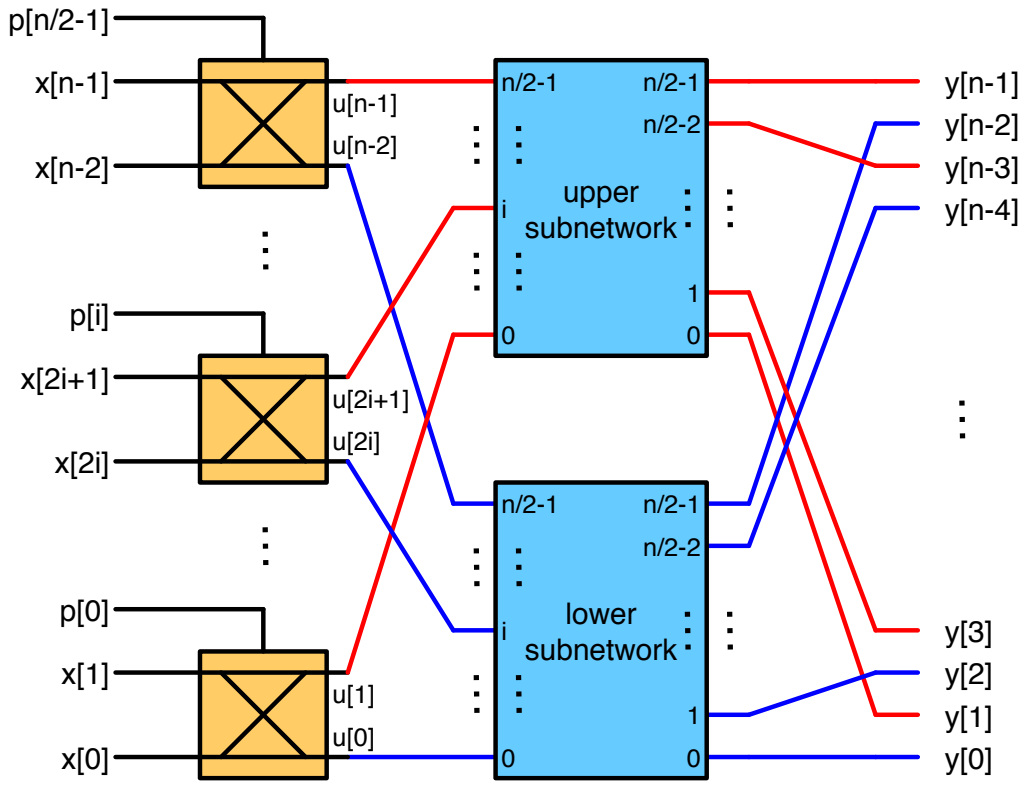
- front-end splitter: partitions valid and invalid inputs
- sequence $z_0, \ldots, z_k$ is not sorted (just the valid inputs)
- only some RBS networks can be used for this construction
- inefficient $O(n)$ algorithm to compute the switch configuration

[2] M.Narasimha. A recursive concentrator structure with applications to self-routing switching networks. *IEEE Transcations on Communications, 42(2-4):896-898, April 1994.*

Two Main Contributions

1. improved Narasimha's configuration circuit with two alternatives:
   - parallel prefix computation of parities (PPC)
     reduces depth from $O(n)$ to $O(\log^2(n))$
   - ranking based computation (RC)
     reduces depth from $O(n)$ to $O(\log(n))$
2. routing partial permutations on general RBS networks
   two general constructions of ternary split modules
   - Split modules as ternary sorters
   - Split modules as ternary concentrators

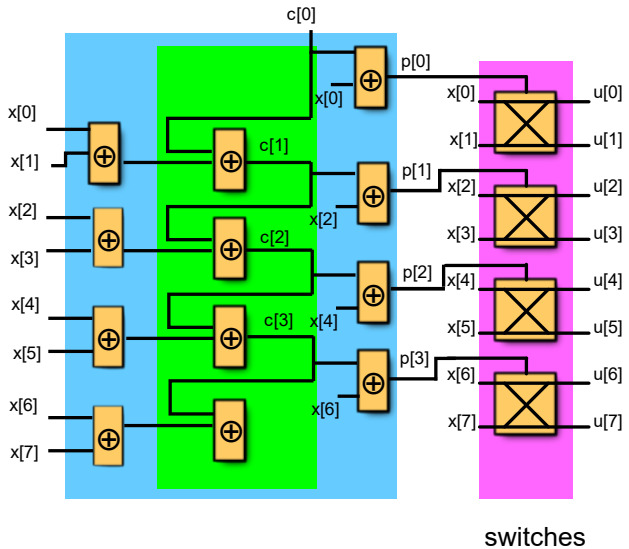
Contribution 1: Consider Narasimha's Splitter



- recursive structure
  - $\frac{n}{2}$ 2×2 crossbar switches
  - lower and upper subnetwork
  - ingoing flip-shuffle, outgoing perfect shuffle permutation
- configuration of switches
  - determine $p_0, \ldots, p_{\frac{n}{2}-1}$
  - inputs $\mathbf{0}/\mathbf{1}$ are evenly distributed to both subnetworks
  - i.e., one to the lower, the next to the upper network
  - subnetworks sort binary inputs

↬ shuffled output y will then be sorted

Sequential Switch Configuration: by Narasimha (1994)



- Narasimha's circuit computes the configuration as follows

$$p_i := \begin{cases} msb(x_0) & : \text{for } i = 0 \\ p_{i-1} \oplus msb(x_{2i-1}) \oplus msb(x_{2i}) & : \text{for } i > 0 \end{cases}$$

- leads to circuit with $O(n)$ XOR gates
- **with a depth (latency) of $O(n)$**

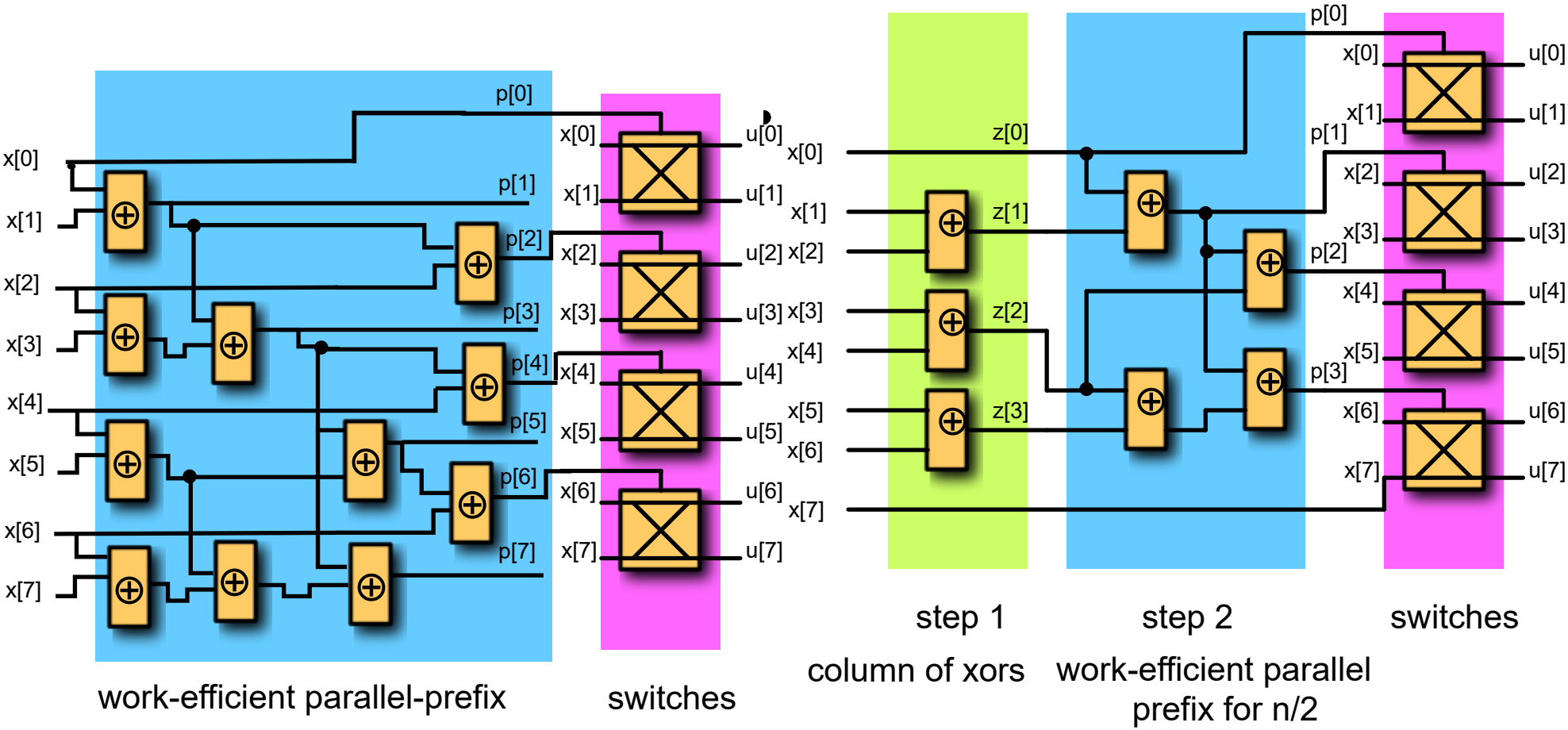
Contribution 1.1: Parallel Parity Configuration (PPC)

- reduces depth from $O(n)$ to $O(\log^2(n))$
- since $\oplus$ is associative, we use a work-efficient parallel-prefix computation, but it computes all prefixes
- **however, we just need half of them**
- **solution: two step procedure**
  - step 1: pairing computes $z_i := x_{2i-1} \oplus x_{2i}$ for $i = 1, \ldots, \frac{n}{2} - 1$
  - step 2: p_i is obtained by a parallel prefix sum of $z_0, \ldots, z_{\frac{n}{2}-1}$:

$$p_i := z_0 \oplus z_1 \oplus \ldots \oplus z_i$$

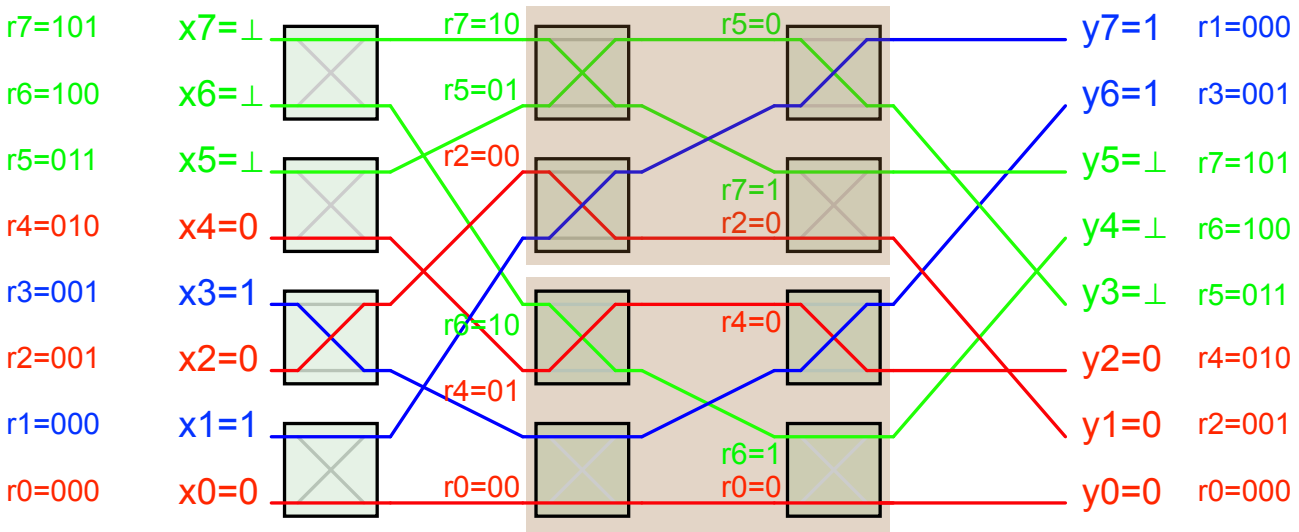
- reduce size from $O(2n)$ to $O(\frac{3}{2}n)$, i.e., by 25%
- almost same circuit size as the sequential configuration, but with improved depth $O(\log^2(n))$

Example: Parallel Parity Configuration (PPC)



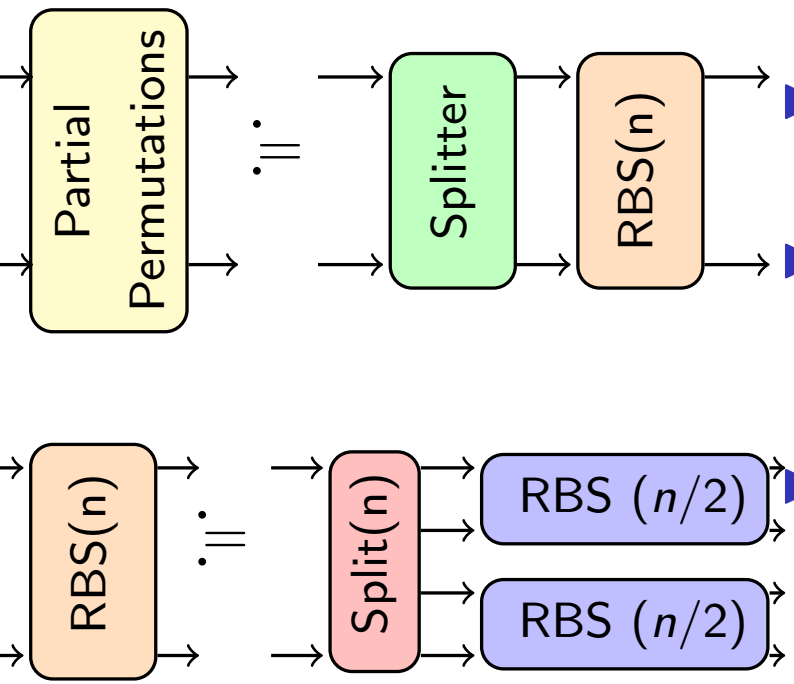
Contribution 1.2:Ranking based Computation (RC)

- reduces depth from $O(n)$ to $O(\log(n))$
- ranks: number of 0s in $x_0, \ldots, x_i$
- ranks: $r_i := \left(\sum_{j=0}^{i} \neg msb(x_i) \right) - 1$



- rank r_i is considered as local target address in that the Split module will route inputs to the outputs

Narasimha's Network: Routing Partial Permutations



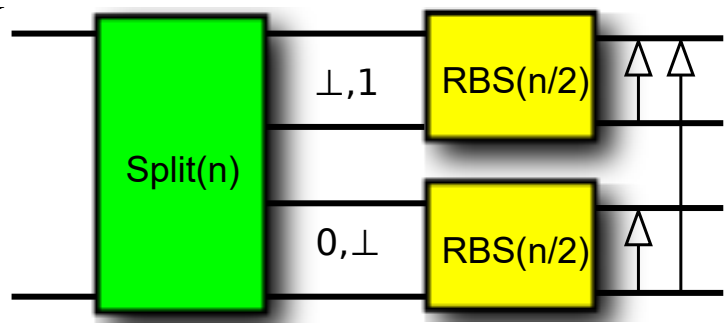
- (front-end) valid splitter:
  - separates valid inputs and invalid inputs
- RBS(n):
  - consists of split module, and two RBS ($n/2$) modules
- Split module:
  - sorts the ternary inputs with ordering $\mathbf{0} \preceq \perp \preceq \mathbf{1}$

Asymptotic Complexities

	sequential	PPC	RC
splitter depth	$O(n)$	$O(\log(n)^2)$	$O(\log(n))$
splitter size	$O(n\log(n))$	$O(n\log(n)^2)$	$O(n\log(n)^2)$
computation	every column	every column	first column
main gates	xor	xor	adder
network depth	$O(n)$	$O(\log(n)^3)$	$O(\log(n)^2)$
network size	$O(n\log(n)^3)$	$O(n\log(n)^3)$	$O(n\log(n)^3)$

- size of the entire network is same for all three circuits
- depth of RC approach is better as compared to others

Contribution 2: Solutions for other RBS networks



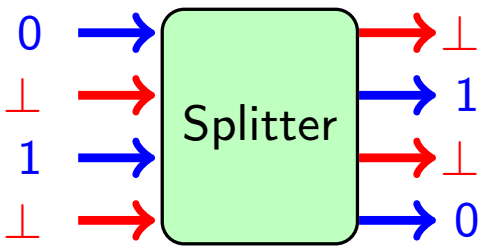
- for partial permutations, we may use ternary or ternary splitters

↬ **1***s* up, **0***s* down, ⊥ anywhere

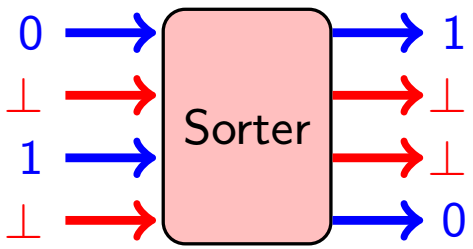
↬ we need ternary sorters or ternary splitters

we proposed three general constructions for ternary split modules based on binary modules, two of them are shown on the next slide

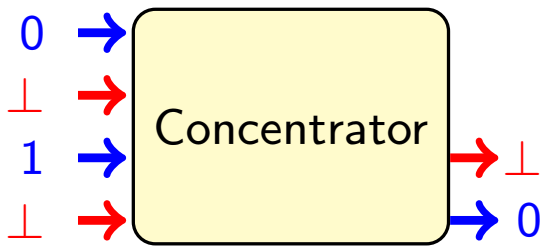
Basic Terminology: Splitter, Sorter, Concentrator



- routes all 0*s* to the lower half and all 1*s* to the upper half



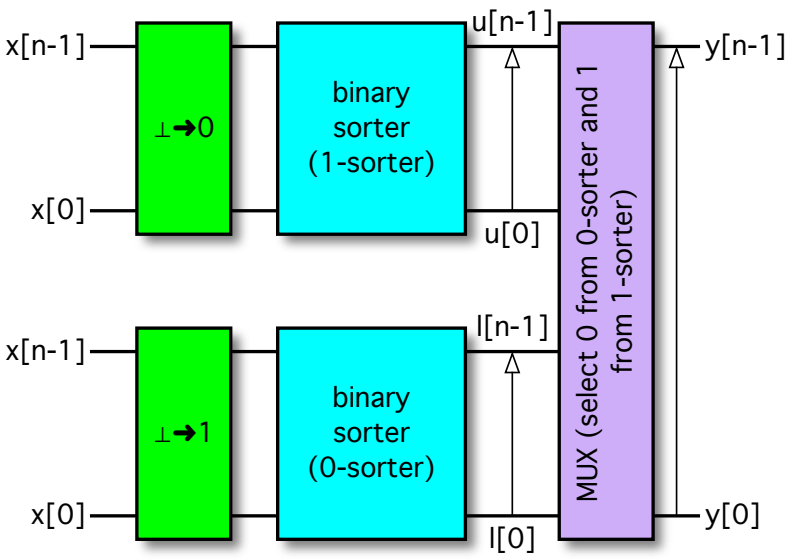
- sorts the sequence with total order $0 \preceq \bot \preceq 1$



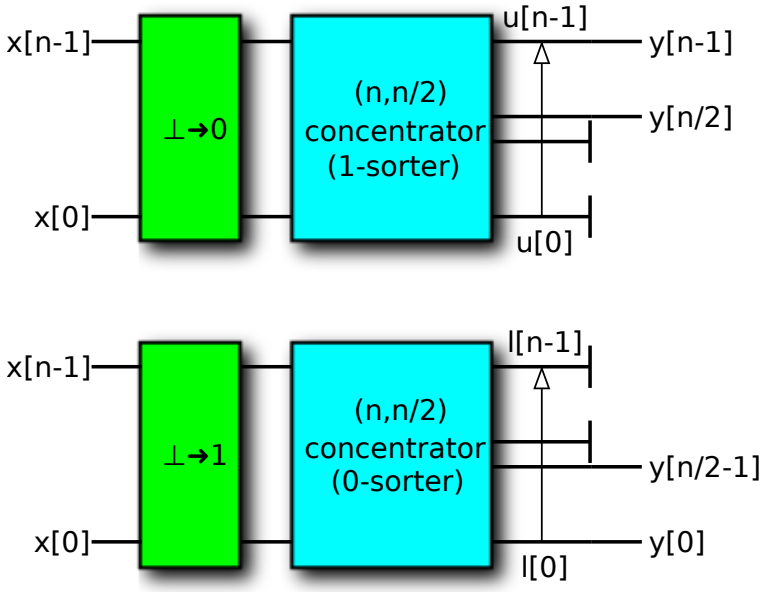
- routes either 0s or 1s to its $\frac{n}{2}$ outputs

  - remark: for (partial) permutations, we have at most $\frac{n}{2}$ 0s and at most $\frac{n}{2}$ 1s in the inputs
  - a splitter can be implemented by two concentrators
  - every sorter is also a splitter, but not vice versa

Contribution 2: Solutions for other RBS networks



- implements a ternary sorter
- doubling circuit size of RBS networks



- implements a ternary splitter
- doubling circuit size of RBS networks

Experimental Results

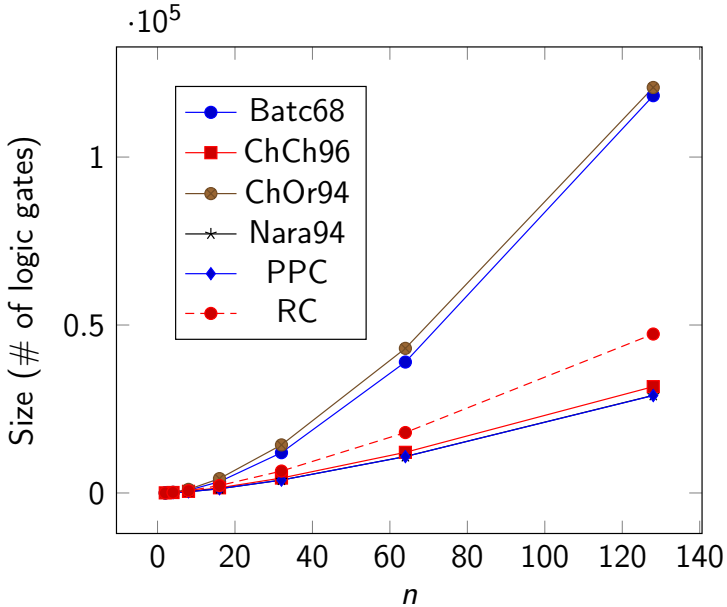
- we consider implementations of four known binary sorters: Batcher-Bitonic [?], Cheng-Chen [?], Chien-Oruç [?], Narasimha [?]
- implemented as a combinational circuit with 1 bit message size
- we compare the circuit size and circuit depth as a measure of the latency (using netlist generator) for different numbers of inputs
- we compare chip area, power consumption and maximal clock frequency (using Cadence 65nm technology) with different numbers of inputs

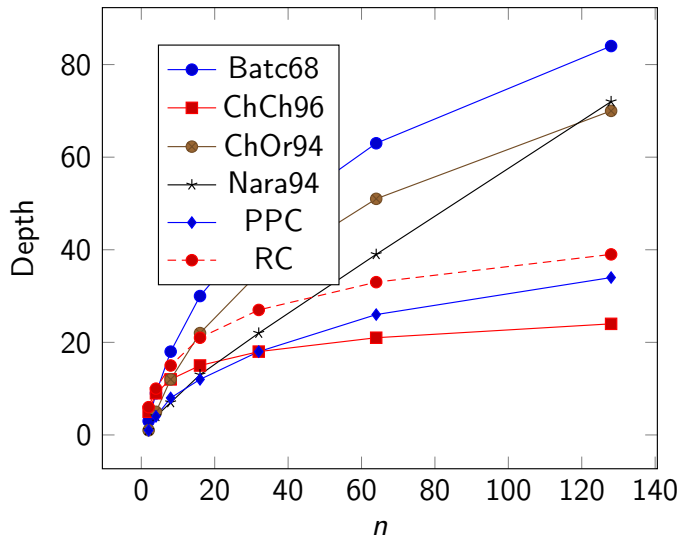
Asymptotic Complexities

network	depth	size
Crossbar	$O(\log(n))$	$O(n^2)$
Narasimha [?]	$O(n)$	$O(n(\log(n))^3)$
PPC	$O(\log(n)^3)$	$O(n(\log(n))^3)$
RC	$O(\log(n)^2)$	$O(n(\log(n))^3)$
Batcher-Bitonic [?]	$O(\log(n)^3)$	$O(n(\log(n))^3)$
Chien-Oruç [?]	$O(\log(n)^3)$	$O(n(\log(n))^3)$
Cheng-Chen [?]	$O(\log(n)^2)$	$O(n(\log(n))^3)$

- RC and Cheng-Chen's network have best depth complexities
- developed networks are as good as so-far known best ones
- but can additionally route partial permutations

Experimental Results - Binary Sorter

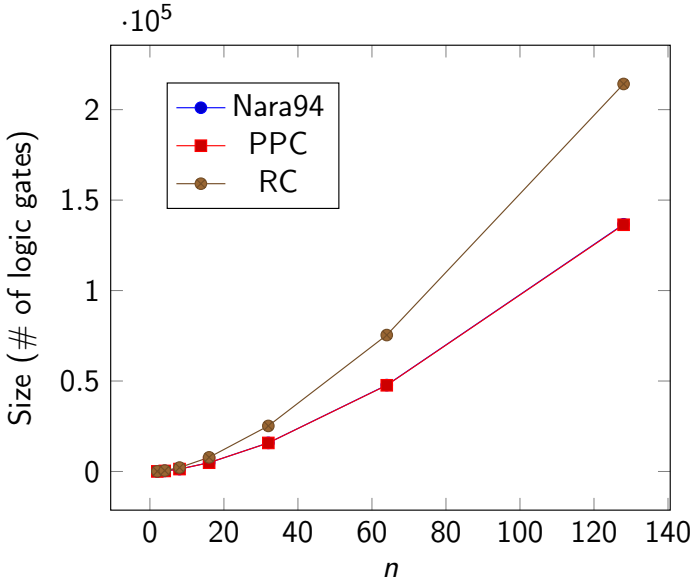


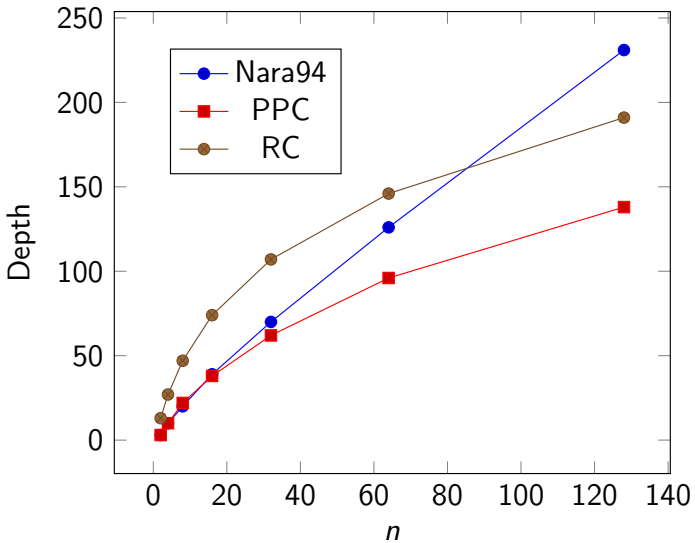


- sizes of ChCh96, Nara94, and PPC are more or less the same
- depths of PPC and the ChCh96 are comparable in the ranges up to $n = 64$,

↬ PPC and Cheng-Chen are the best for practical depth and size

Experimental Results - Narashima's RBS Network with Front-End Splitter





- sizes of PPC and Nara94 is almost the same
- depth of PPC is good

↪ again, PPC is the best for practical depth and size

Summary

- system-on-a-chip designs require efficient interconnection networks
- mesh networks are not powerful enough
- crossbars lead to large circuits
- sorting networks are attractive alternatives, but cannot handle invalid inputs
- we considered extensions of sorting networks for partial permutations
  - using ternary sorters as Split modules in RBS networks
  - using prefix sequences generated by a front-end splitter
  - the latter requires prefix-based configurations, and developed two efficient versions (parallel prefix and ranking based configurations)

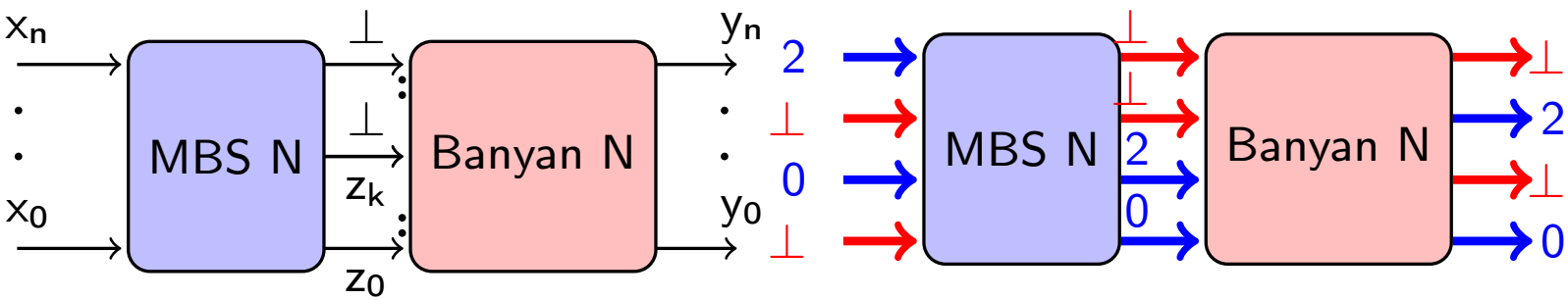
The End

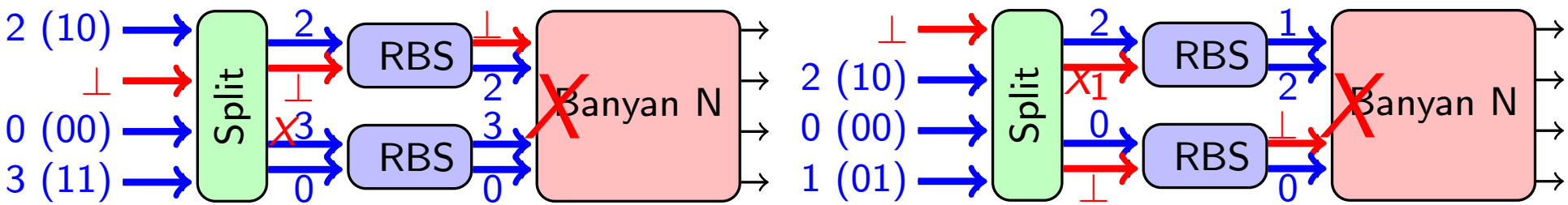
Thank you for your attention.

Questions?

References I

RBS-Banyan Networks



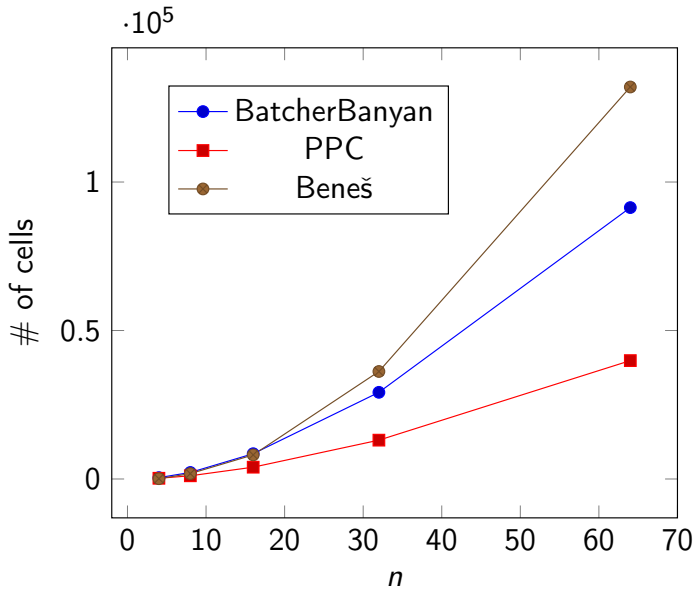


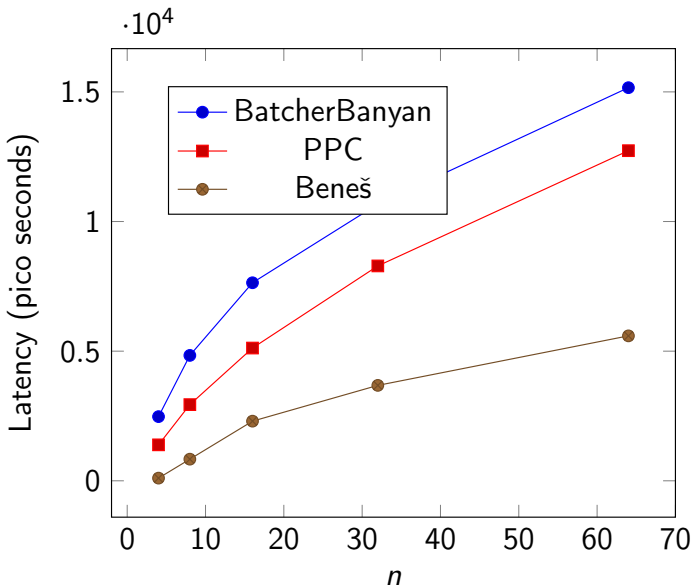
Asymptotic Complexities

	depth	size
Merge module	$O(\log(n))$	$O(n(\log(n))^2)$
Split module (permutation)	$O(\log(n))$	$O(n(\log(n))^2)$
Split module (concentrator)	$O(\log(n))$	$O(n)$

networks	depth	size
MBS	$O(\log^2(n))$	$O(n\log^3(n))$
RBS	$O(\log^2(n))$	$O(n\log^3(n))$
RBS (concentrator)	$O(\log^2(n))$	$O(n\log(n))$
AKS	$O(\log(n))$	$O(n\log(n))$

Experimental Results - Other Networks





- latency of Beneš is best
- PPC is the best for practical size