Energy and Memory Efficient Mapping of Bitonic Sorting on FPGA

Ren Chen, Sruja Siriyal, Viktor K. Prasanna

Ming Hsieh Department of Electrical Engineering

University of Southern California

Ganges.usc.edu/wiki/TAPAS

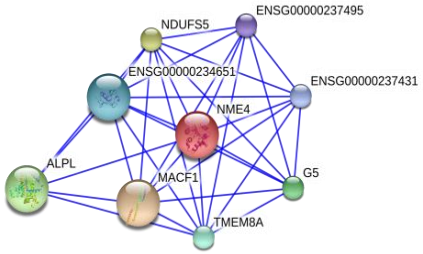
Outline

- Introduction
- Background and Related Work
- Memory and Energy Efficient Mapping
- Architecture Implementation
- Experimental Results
- Conclusion and Future Work

Applications of Sorting Algorithm

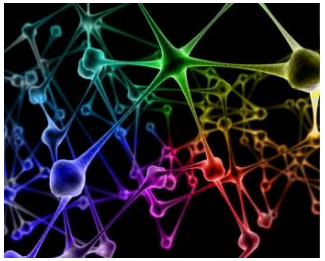
Online social networks

Protein interactions

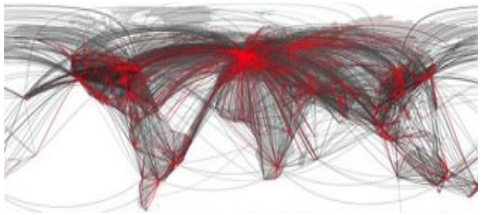


WWW

Neural network



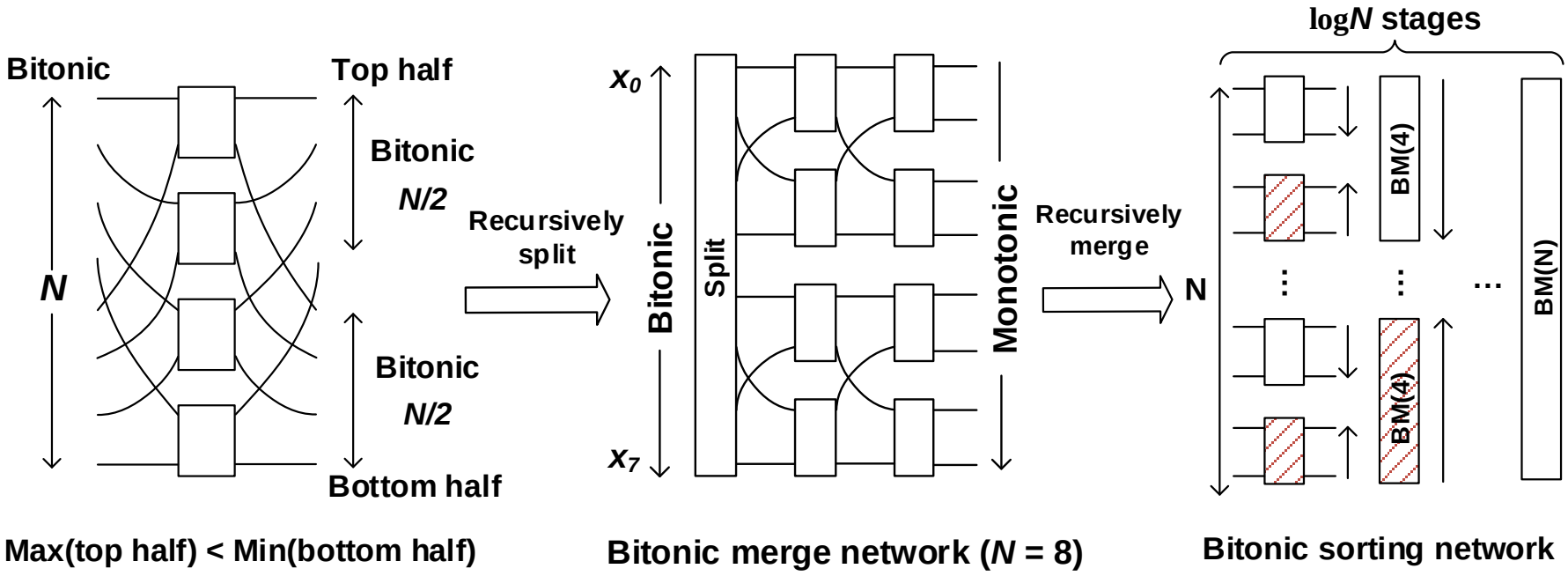
Air traffic network



Citation networks

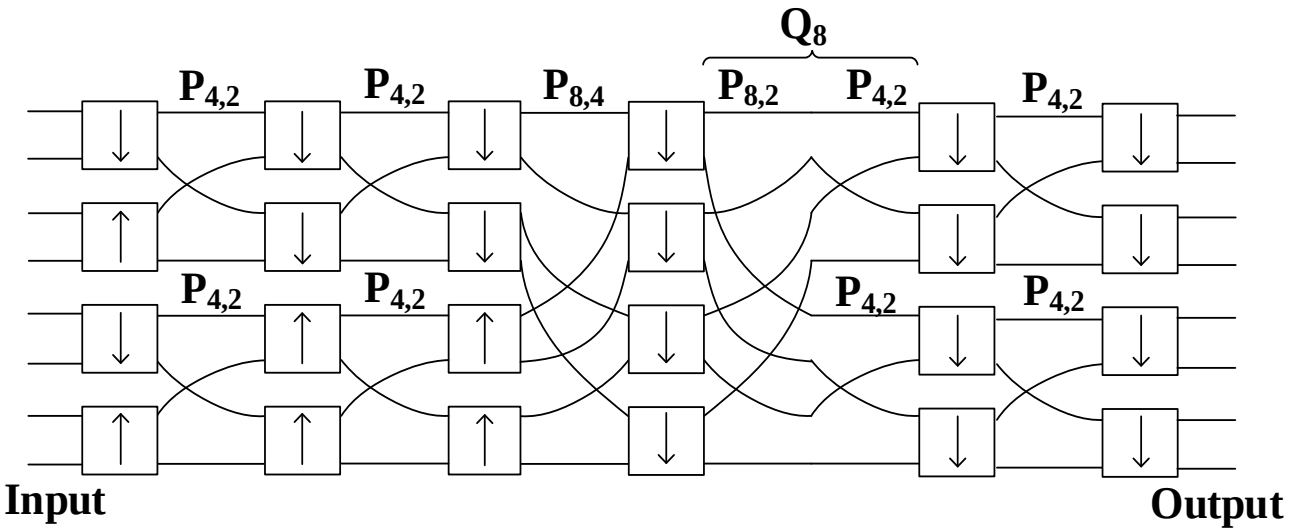
Bitonic Sorting Network (BSN) (1)

- Parallel sorting network with simple control scheme
- Hardware implementation can achieve extremely high throughput
- Computation complexity: $O(N \log^2 N)$ for problem size N



BSN (2)

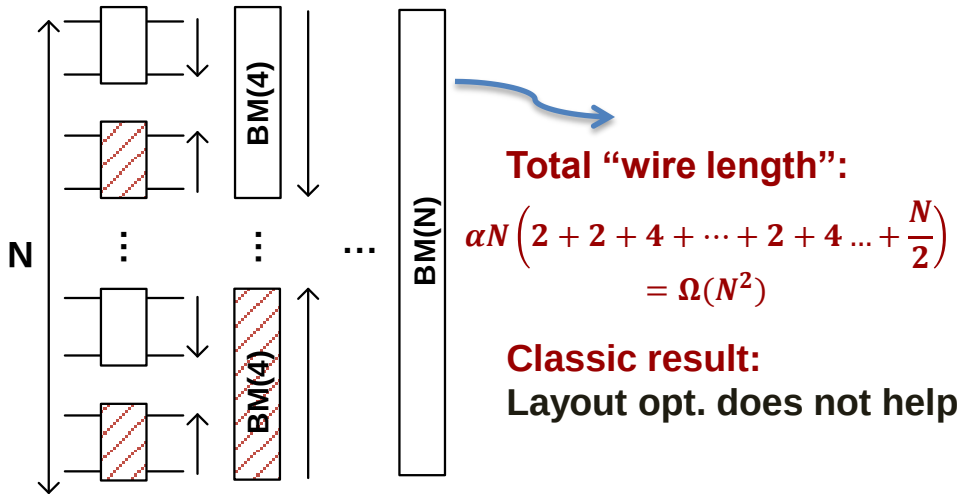
- Comparison stages: $\log N\,(\log N + 1)/2$
- Communication stages: $\log N\,(\log N + 1)/2$
  - Inter-stage communication → data permutation between comparison stages
  - Two types of permutation patterns: stride permutation $P_{m,t}$, $Q_m = \left(I_2 \otimes P_{m/2,m/4}\right)$
  - $2\log N$ unique permutation patterns



8-input bitonic sorting network

BSN (3)

- Resource consumption: $O(N \log^2 N)$ compare-and-swap (CAS) units
- Total "wire length": $\Omega(N^2)$
  - "Wire length": data communication distance, **communication power** $\propto$ "wire length"
- Key issue: communication power consumption between adjacent stages

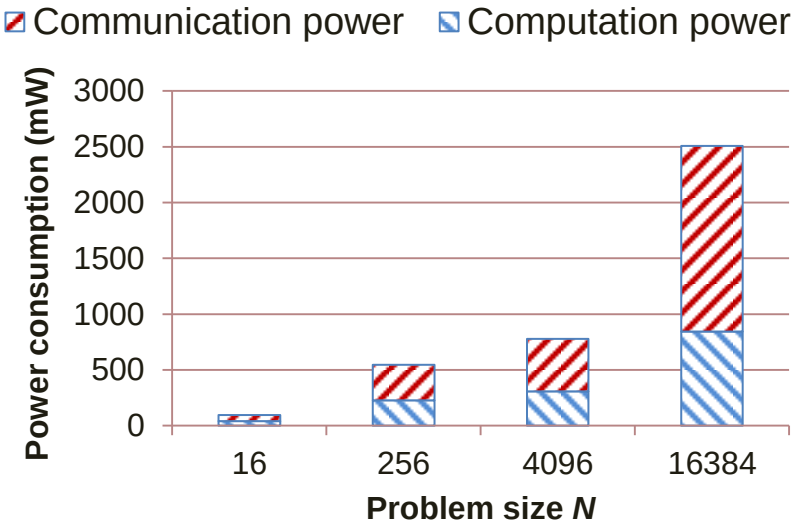


Bitonic sorting network

Total "wire length":

$$\alpha N\left(2+2+4+\cdots+2+4\ldots+\frac{N}{2}\right) = \Omega(N^2)$$

Classic result:
Layout opt. does not help

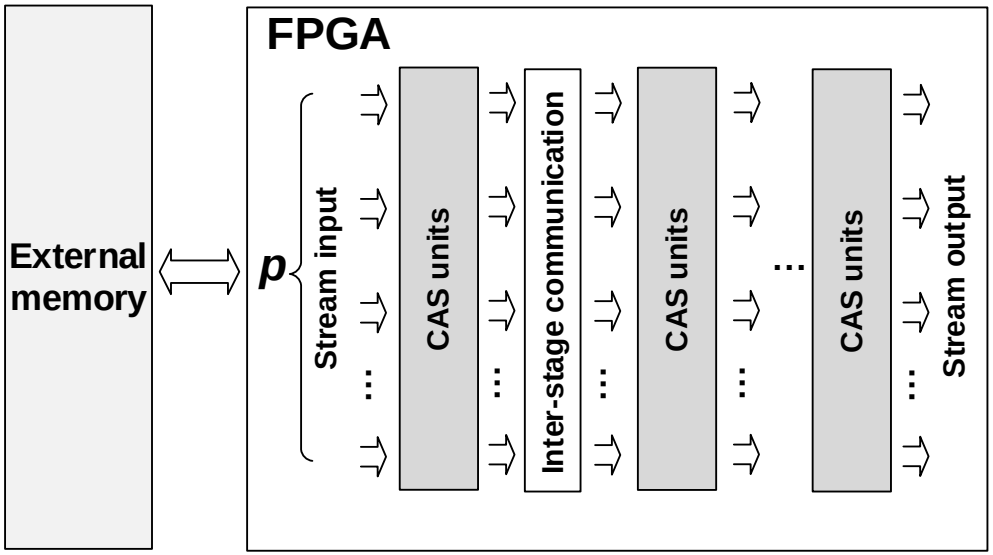


Experimental results on Virtex-7 FPGA

Problem Definition (1)

Sort streaming data with a fixed data parallelism

- Data memory: stores the input consisting of N-key data sequences
- Input/output: in a streaming manner and at a fixed rate
- Data parallelism p: # of keys processed each cycle per comparison stage
- Inter-stage communication: data permutation between adjacent stages



Problem Definition (2)

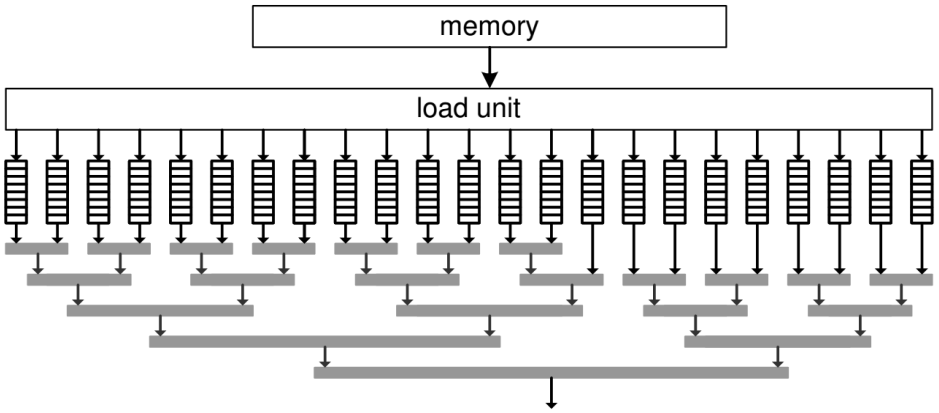
- Performance metrics
  - Throughput
    - Defined as the number of bits sorted per second (Gbits/s)
    - Product of number of keys sorted per second and data width per key
  - Energy efficiency
    - Defined as the number of bits sorted per unit energy consumption (Gbits/Joule)
    - Calculated as the throughput divided by the average power consumption
  - Memory efficiency
    - Throughput achieved divided by the amount of on-chip memory used by the design (in bits)

Outline

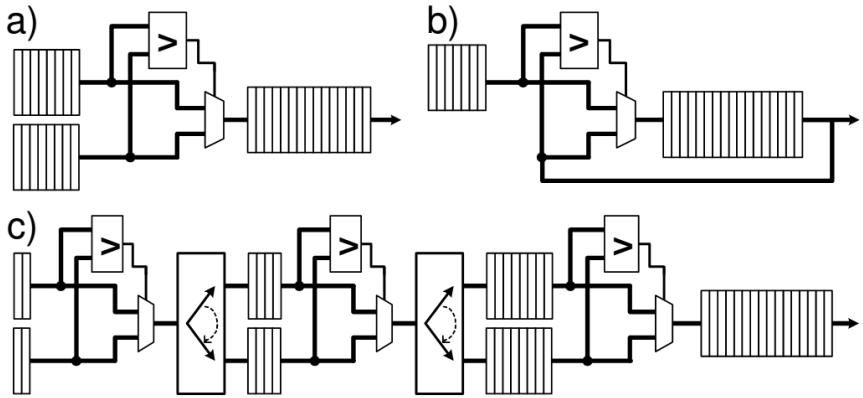
- Introduction
- Background and Related Work
- Memory and Energy Efficient Mapping
- Architecture Implementation
- Experimental Results
- Conclusion and Future Work

Related Work (FPGA '11, D. Koch and J. Torresen)

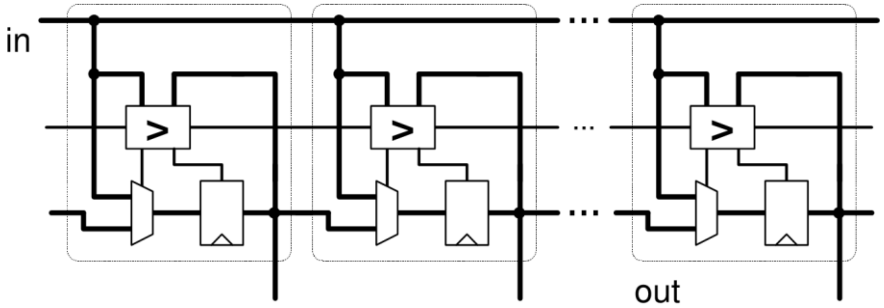
- Tree merge sorter
- FIFO-based merge sorter
- Insertion sorter
- FIFO & Tree
- 2 GB/s using on-chip memory



Tree merge sorter



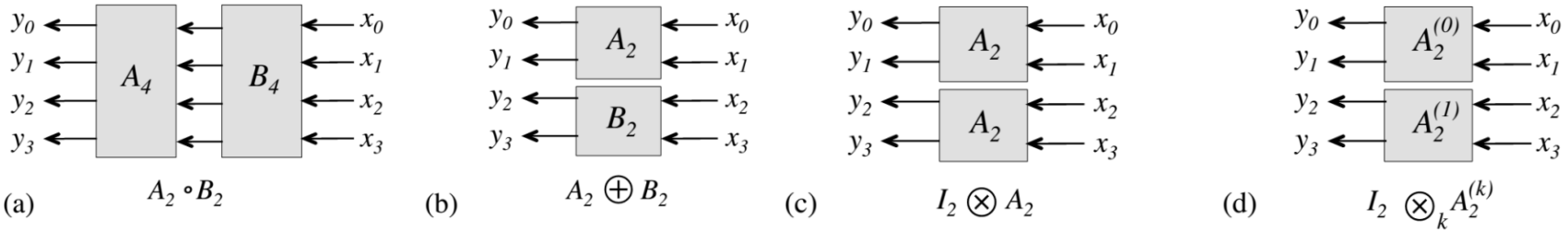
FIFO-based merge sorter



Insertion sorter

Related Work (DAC '12, M. Zuluaga and M. Püschel)

- Hardware generator for sorting
- Domain-specific language based



SN1: $$\prod_{i=1}^{t-1}\left[(I_{2^{t-1}}\otimes S_2)\prod_{j=2}^{t-i+1}\left[\left(I_{2^{t-j}}\otimes(I_2\otimes L_{2^{j-1}}^{2^{j-2}})L_2^{2^j}\right)(I_{2^{t-1}}\otimes S_2)\right]\left(I_{2^{i-1}}\otimes\left(L_{2^{t-i+1}}^{2^{t-i}}(L_{2^{t-i}}\otimes J_{2^{t-i}})\right)\right)\right](I_{2^{t-1}}\otimes S_2)$$

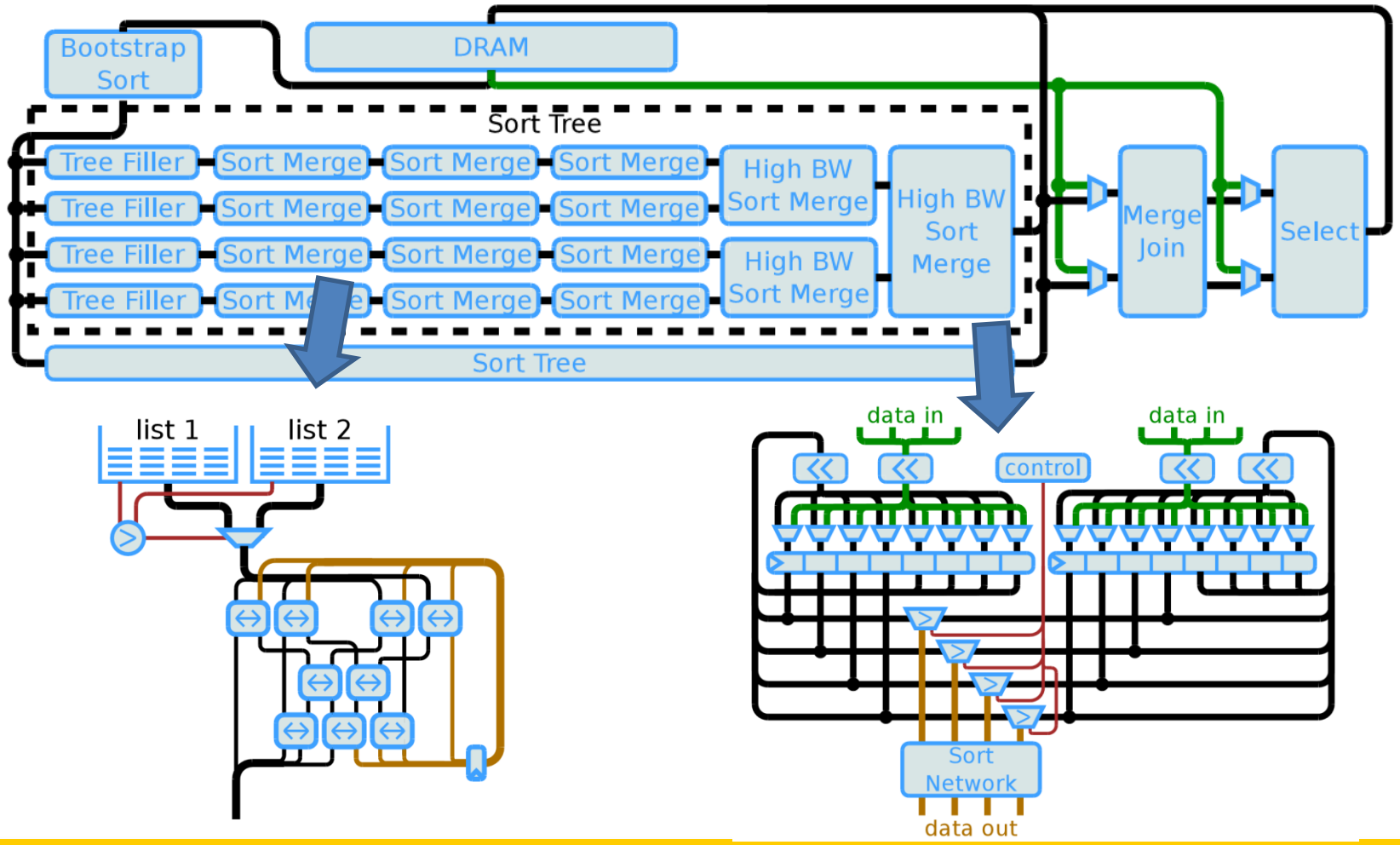
SN2: $$\prod_{i=1}^{t-1}\left[\prod_{j=2}^{t-i+1}\left[(I_{2^{t-1}}\otimes S_2)(I_{2^{i-1}}\otimes L_{2^{t-i}}^{2^{t-i+1}})\right](I_{2^{i-1}}\otimes(I_{2^{t-i}}\otimes J_{2^{t-i}}))\right](I_{2^{t-1}}\otimes S_2)$$

SN3: $$\prod_{i=1}^{t-1}\left[(I_{2^{t-1}}\otimes_m X_2^{g(i,m)})\prod_{j=2}^{t-i+1}\left[\left(I_{2^{t-j}}\otimes(I_2\otimes L_{2^{j-1}}^{2^{j-2}})L_2^{2^j}\right)(I_{2^{t-1}}\otimes_m X_2^{g(i,m)})\right](I_{2^{i-1}}\otimes L_{2^{t-i+1}}^{2^{t-i}})\right](I_{2^{t-1}}\otimes_m X_2^{g(i,m)})$$

SN4: $$\prod_{i=1}^{t-1}\left[\prod_{j=2}^{t-i+1}\left[(I_{2^{t-1}}\otimes_m X_2^{g(i,m)})(I_{2^{i-1}}\otimes L_{2^{t-i+1}}^{2^{t-i}})\right]\right](I_{2^{t-1}}\otimes_m X_2^{g(t,m)});\quad g(i,m)=\begin{cases}1, & m[t-i]=1 \text{ and } i\neq 1\\ 2, & (m[t-i]=0 \text{ or } i=1\end{cases}$$

SN5: $$\prod_{i=0}^{t-1}\prod_{j=0}^{t-1}\left[(I_{2^{t-1}}\otimes_m X_2^{f(i,j,m)})L_{2^t}^{2^{t-1}}\right];\quad f(i,j,m)=\begin{cases}0, & t-1<j+i\\ 1, & m[t-1-j-i]=1 \text{ and } i\neq 0\\ 2, & m[t-1-j-i]=0 \text{ or } i=0\end{cases}$$

Related Work (FPGA ’14, J. Casper and K. Olukotun)



Memory and Energy Efficient Mapping

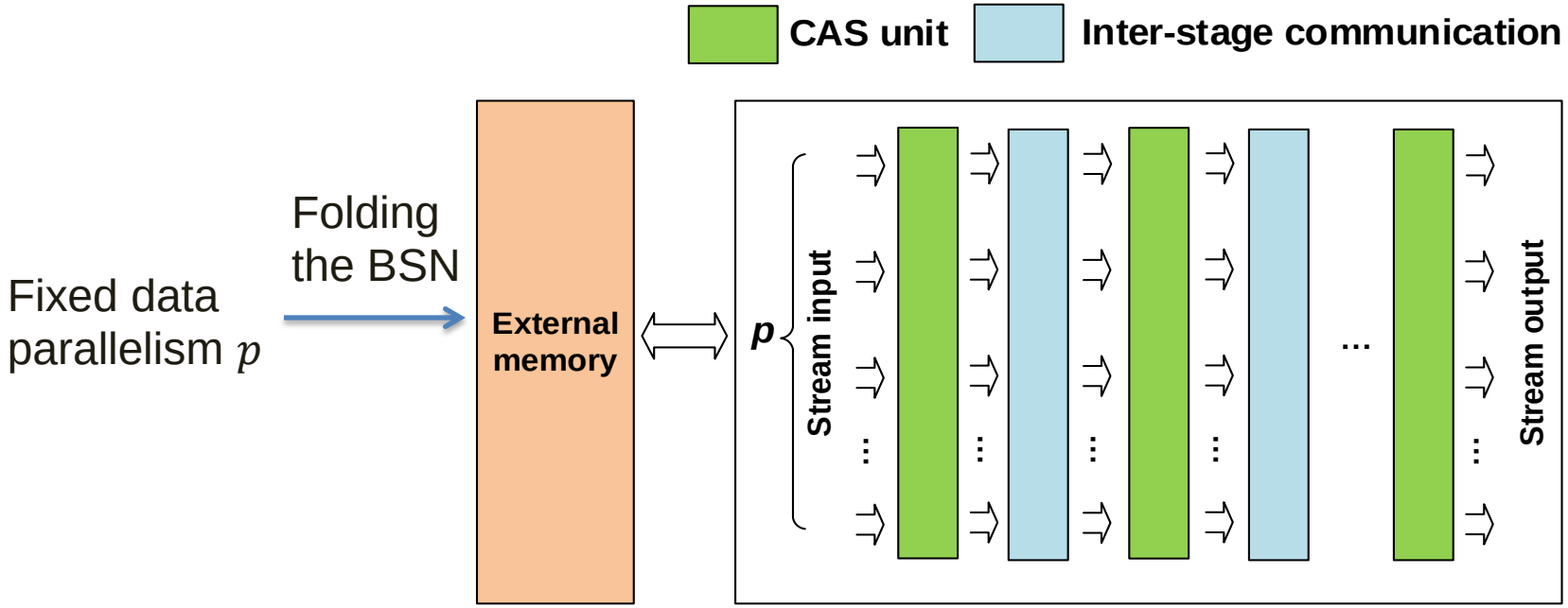
- Drawbacks of the state-of-the-art
  - High throughput not guaranteed
  - Design scalability needs to be improved
  - No analysis provided
  - Data parallelism is limited
- We propose a mapping approach to obtain a streaming sorting architecture
  - BSN based
  - Utilizes Clos network for inter-stage communication
  - Highly optimized wrt. energy efficiency
  - Achieves optimal memory efficiency ($O(p/N)$)
  - Scalable with N and p
  - Supports processing continuous data streams

Outline

- Introduction
- Background and Related Work
- Memory and Energy Efficient Mapping
- Architecture Implementation
- Experimental Results
- Conclusion and Future Work

Proposed Mapping Approach

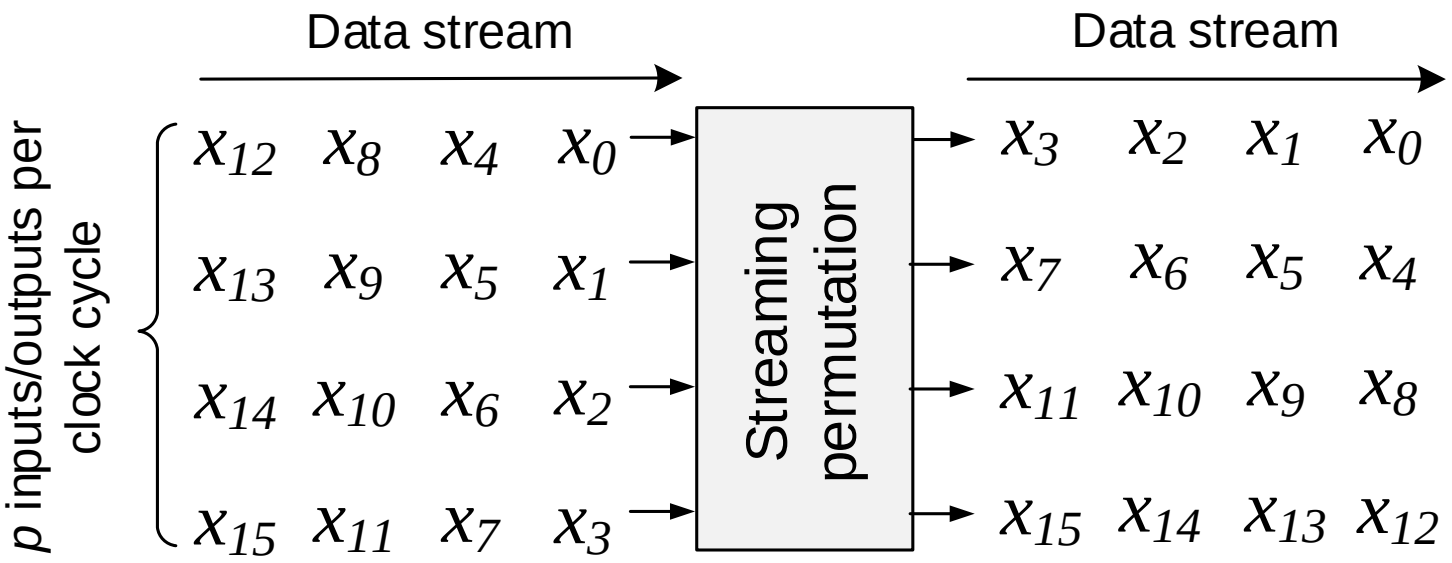
- Fold the BSN for a fixed data parallelism p
- Fold the Clos network to perform inter-stage communication
- Support continuous data streams to maximize throughput



Streaming architecture for sorting

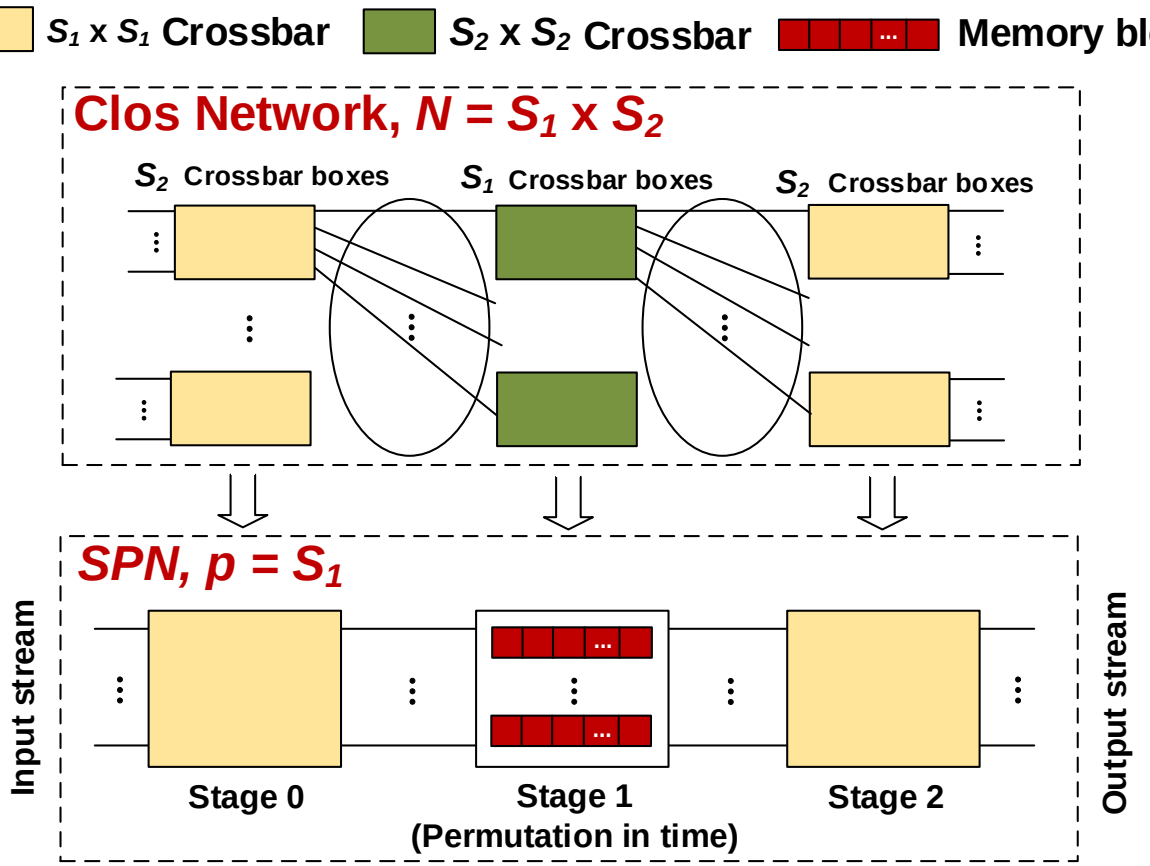
Mapping the Clos Network (1)

- Fold the BSN → Streaming permutation



Mapping the Clos Network (2)

- Fold the Clos network into a 3-stage Streaming Permutation Network (*SPN*)
- *SPN*
  - Stage 0
    - S_1-*to*-S_1 connection
  - Stage 1
    - S_1 single-port memory blocks, each of size S_2
  - Stage 2
    - S_1-*to*-S_1 connection
- Permutation in time
  - Permuting temporal order of data elements



Mapping the Clos Network (3)

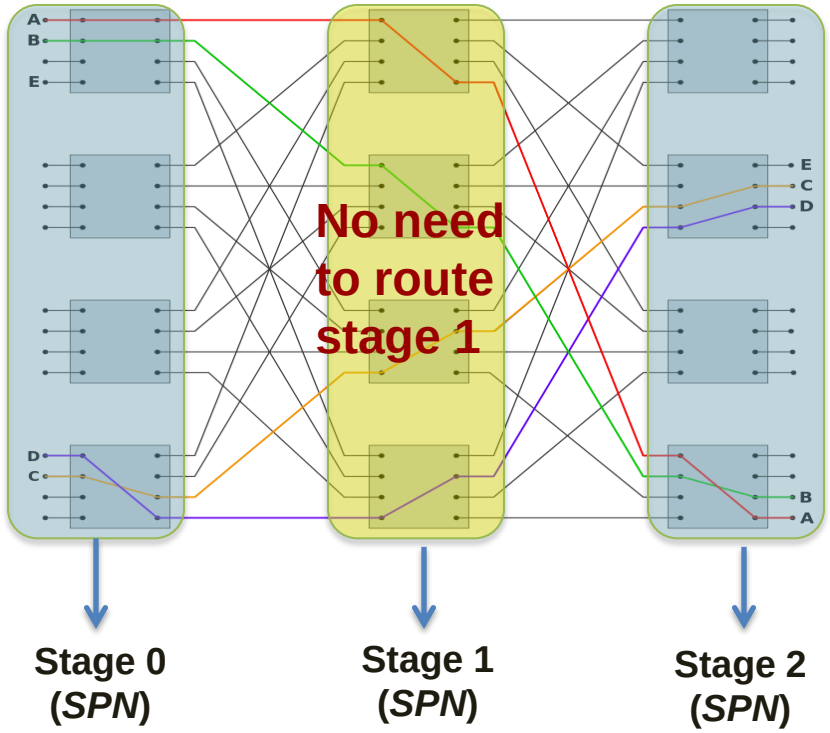
Theorem 1: With $p = S_1$, the proposed *SPN* can realize any given permutation on streaming input of an N-key data sequence without any memory conflicts using S_1 single-port memory blocks, each of size S_2 ($S_2 \geq S_1$).

Memory conflict: occurs if concurrent read or write access to more than one word in a single-port memory block is performed in a clock cycle.

Mapping the Clos Network (4)

Theorem 2: For any given permutation, time to "route" *SPN* is $O(N \log p)$ $(1 \leq p \leq N)$

- "Routing" for *SPN*
  - Obtain control bits for stage 0 and stage 2
  - Obtain memory addresses for stage 1
  - Configured dynamically or statically
- "Routing" for $N = 4096, p = 64$
  - Time in state-of-the-art [1]: 6 minutes
  - Time for the proposed *SPN*: 16 s
  - **22x improvement**



[1] P. A. Milder, J. C. Hoe., M. Puschel. "Automatic generation of streaming datapaths for arbitrary fixed permutations." IEEE DATE, pp. 1118-1123., 2009.

Mapping the Clos Network (5)

- Key advantages of utilizing Clos network:
  - Result in efficient control logic for *SPN*
    - $O(\log \frac{N}{p})$ control logic
  - Run-time "programmability"
    - Time multiplexing CAS units to save logic
    - Low "programming" overhead
  - Scalable wrt. N and p
    - $O(p \log p)$ logic consumption for stage 0 and stage 2
    - p single-port memory blocks

Memory Efficient Permutation in Time (1)

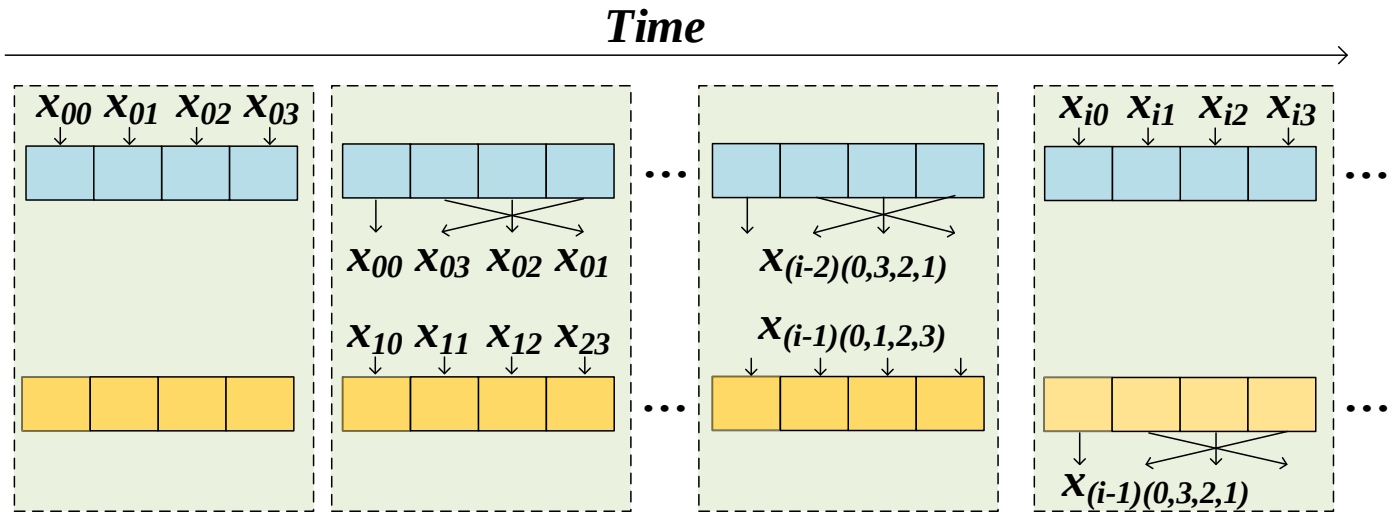
- Memory power for permutation in time (stage 1 in *SPN*) is significant, especially when permuting continuous data streams

- Dual-port memory:
  - One read port and one write port, independently using different memory addresses
  - E.g. 36kb BRAM on Xilinx Virtex-7 in the simple dual-port mode

- "Single-port" memory
  - One port supports read-before-write operation to the same memory location in one cycle
  - Use a single memory address port
  - E.g. 18kb BRAM on Xilinx Virtex-7 in the single-port mode

Memory Efficient Permutation in Time (2)

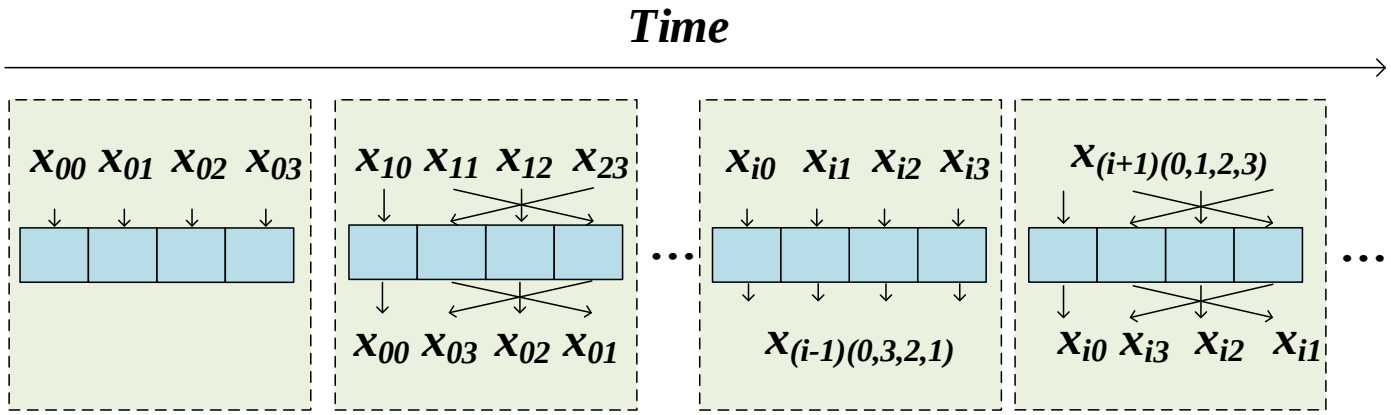
- Permutation in time on continuous data streams can be performed using dual-port memory (state-of-the-art):
  - Store data using $2S_2$ memory for permuting S_2-key data sequences
  - Store addresses using $O(S_2)$ memory: 0,3,2,1 are stored in the example below



Permutation in time using dual-port memory

Memory Efficient Permutation in Time (3)

- We develop an in-place algorithm for permutation in time on continuous data streams:
  - Store data in single-port memory with size of S_2
  - Update memory addresses by an address generation unit using $O(\log S_2)$ logic

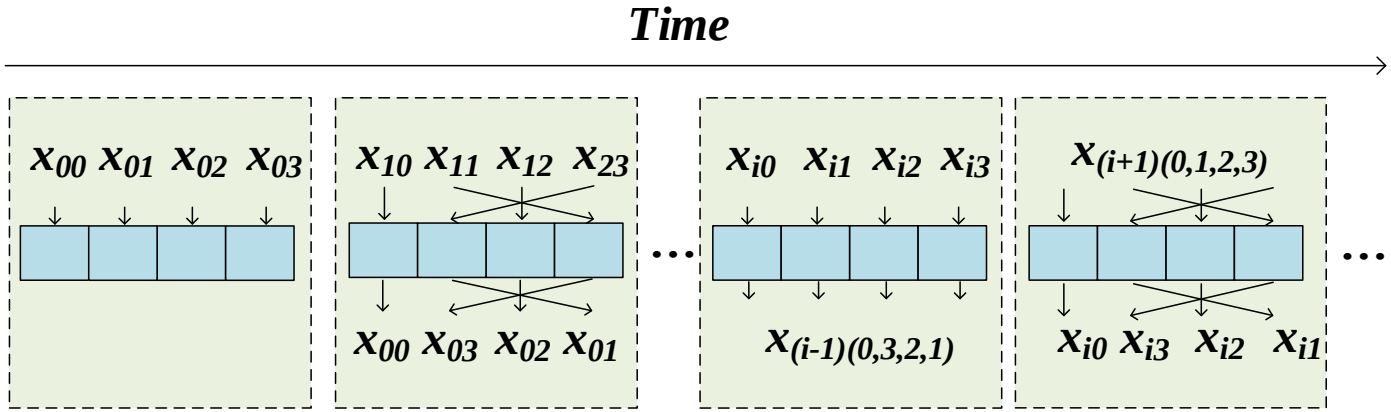


Permutation in time using single-port memory

Memory Efficient Permutation in Time (4)

Theorem 3: Any permutation in time on continuous data streams consisting of S_2-element sequences can be realized using a single-port memory of size S_2

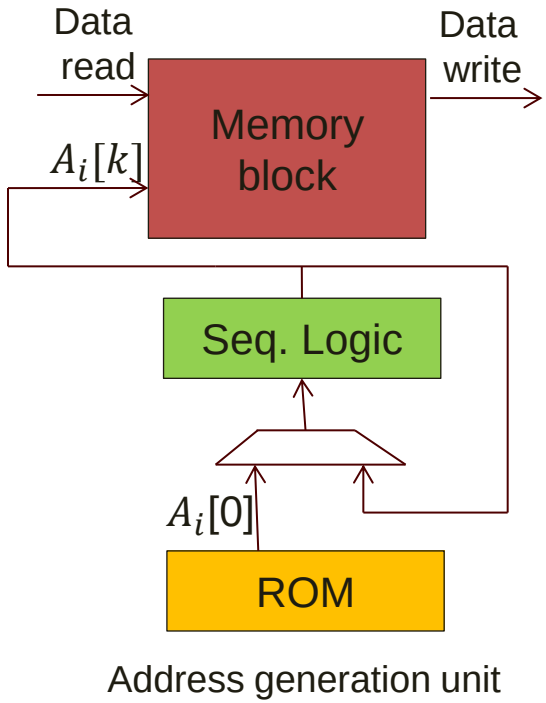
- Active memory address ports reduced by 50%
- Memory size reduced by 50%



Permutation in time using single-port memory

Memory Efficient Permutation in Time (5)

- Implementation of in-place algorithm
  - Use matrix P to represent the **permutation in time**
  - Given **address sequence** $A_0 = \{0,1,2,\dots,S_2-1\}^T$
  - Assume A_{i-1} is used for ith permutation in time, thus $A_i = PA_{i-1}$
  - During ith permutation, in cycle j, $A_i[k]$ is the memory address
  - Based on [2], there always exists a constant q, such that $P^{q+1} = I$, thus $PA_q = P^2A_{q-1} = \cdots = P^{q+1}A_0 = A_0$
  - For stride permutation, $q = \log S_2$



Address generation unit

Theorem 4: For any given permutation in time, the proposed in-place algorithm requires a constant number of address sequences.

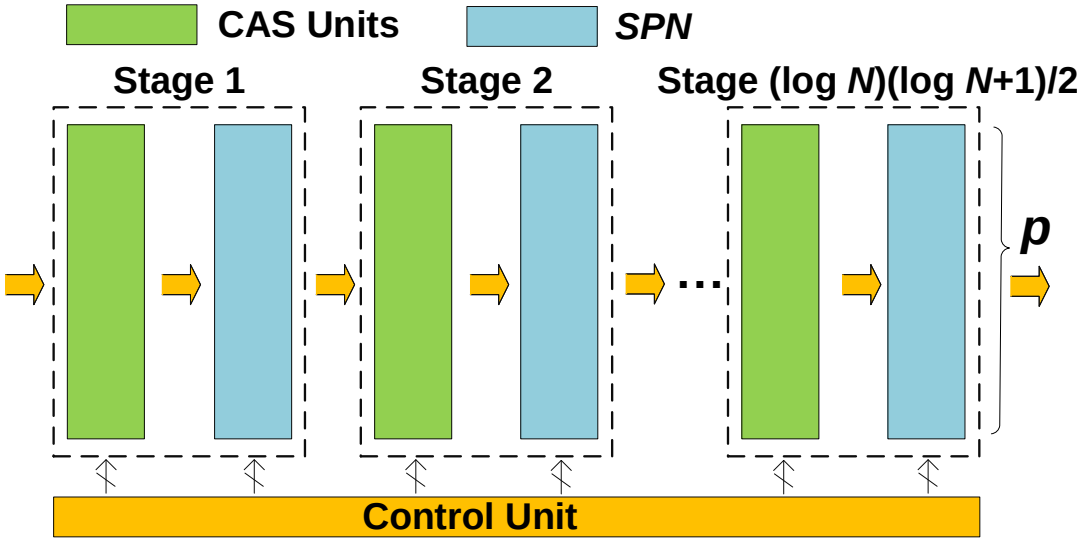
[2] R. A. Brualdi. Combinatorial matrix classes, volume 13. Cambridge University Press, 2006.

Outline

- Introduction
- Background and Related Work
- Memory and Energy Efficient Mapping
- Architecture Implementation
- Experimental Results
- Conclusion and Future Work

Architecture Implementation (1)

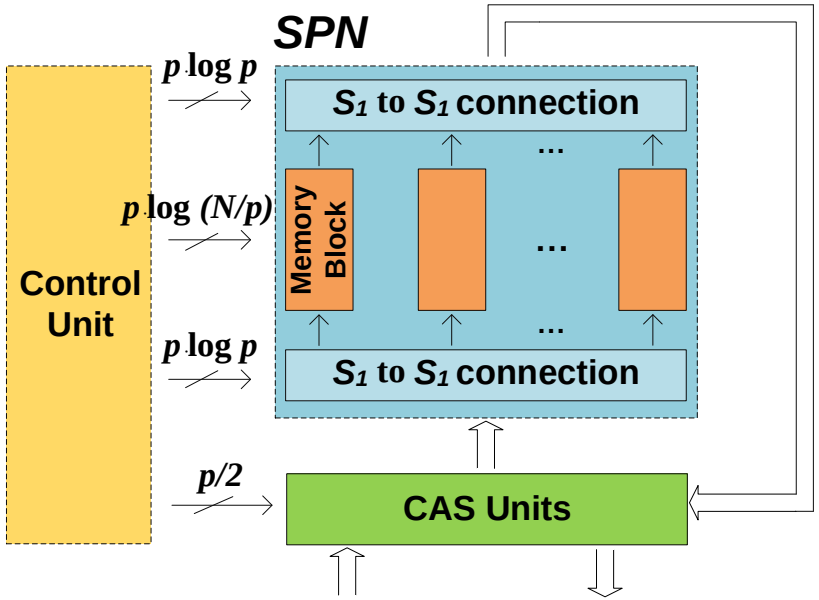
- High Throughput (HT) Design
  - Supports sorting continuous data streams
  - Overall latency: $6N/p + o(N/p)$ $(1 \leq p \leq N/\log^2 N)$
  - Fully pipelined and can maximize the I/O bandwidth utilization



High Throughput Design

Architecture Implementation (2)

- Resource Efficient Design
  - Trade off throughput for area for large scale sorting
  - Use "programmable" *SPN* to time multiplex $p/2$ CAS units
  - "Program" the *SPN* to perform $2\log N$ unique permutation patterns



Resource Efficient Design

Architecture Implementation (3)

Design	Latency	Logic	Memory	Memory type	Through-put	**Memory/ throughput**
FPGA '14	$o(N \log p\,/p)$	$o(p \log N)$	$o(N)$	n/a	$o(p/\log p)$	$2N + o(N)$
DAC '12	$\frac{6N}{p} + o(N/p)$	$o(p \log^2 N)$	$12N + o(N)$	Dual-port	$o(p)$	$\frac{12N}{p} + o(N/p)$
FPGA '11	$o(N)$	$o(\log N)$	$2N + o(N)$	Dual-port	$o(1)$	$2N + o(N)$
TC '00	$o(\frac{N \log N}{p \log p})$	$o(p \log^2 p)$	$o(N)$	Dual-port	$o(\frac{p \log p}{\log N})$	$o(\frac{N \log N}{p \log p})$
HT Design	$\boldsymbol{\frac{6N}{p} + o(N/p)}$	$o(p \log^2 N)$	$6N + o(N)$	**Single-port**	$o(p)$	$\boldsymbol{\frac{6N}{p} + o(N/p)}$
Resource Efficient Design	$o(N \log^2 N\,/p)$	$o(p)$	$o(N)$	Dual-port	$o(\frac{p}{\log^2 p})$	$o(N \log^2 N\,/p)$

Outline

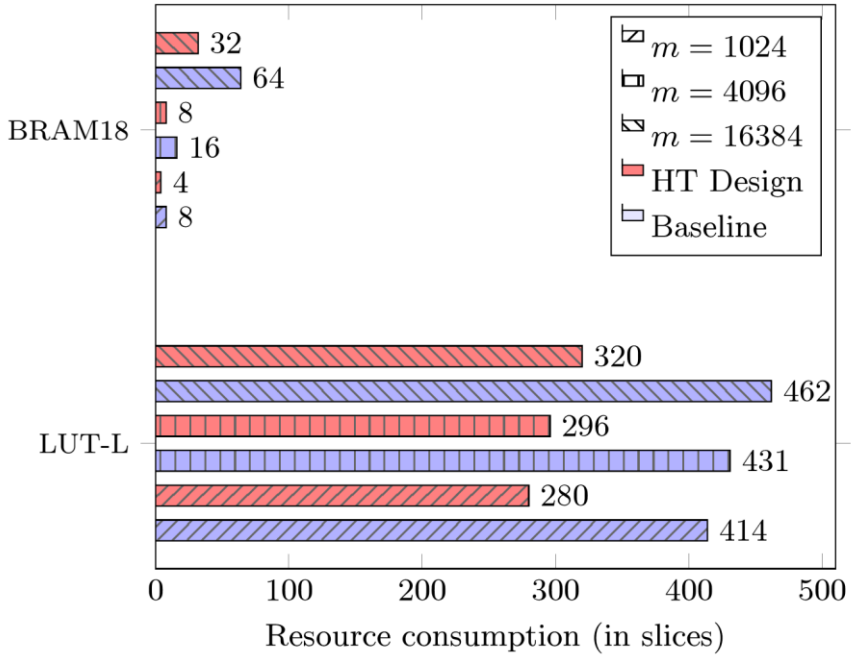
- Introduction
- Background and Related Work
- Memory and Energy Efficient Mapping
- Architecture Implementation
- Experimental Results
- Conclusion and Future Work

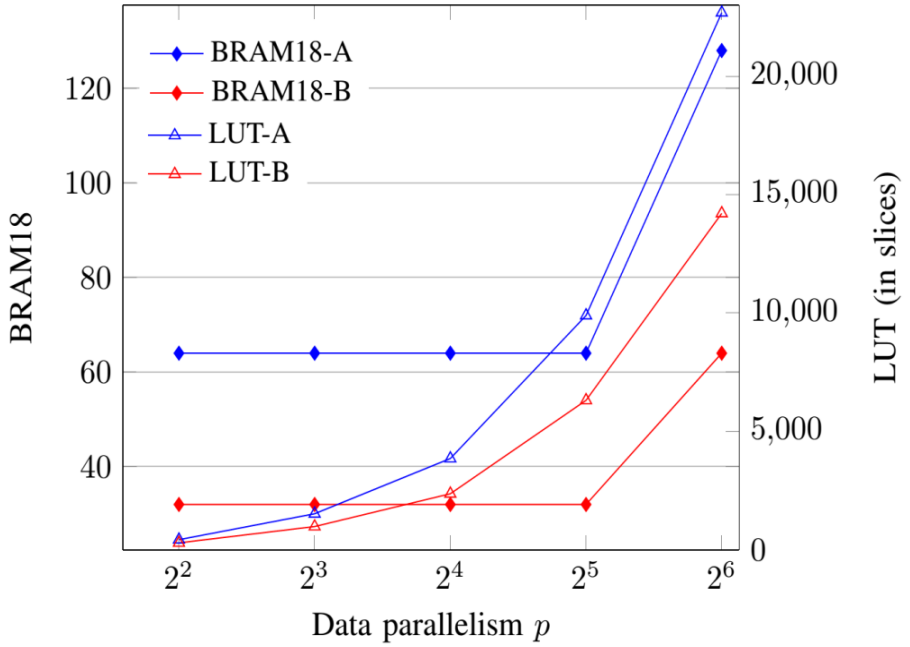
Experimental Setup

- Platform and tools
  - Xilinx Virtex-7 XC7VX980T , speed grade -2L
  - Xilinx Vivado 2014.2 and Vivado Power Analyzer
- Input vectors for simulation
  - Randomly generated with an average toggle rate of 50% (pessimistic estimation)
- Performance metrics
  - **Throughput**
  - **Energy efficiency**
  - **Memory efficiency**

Experimental Results (1)

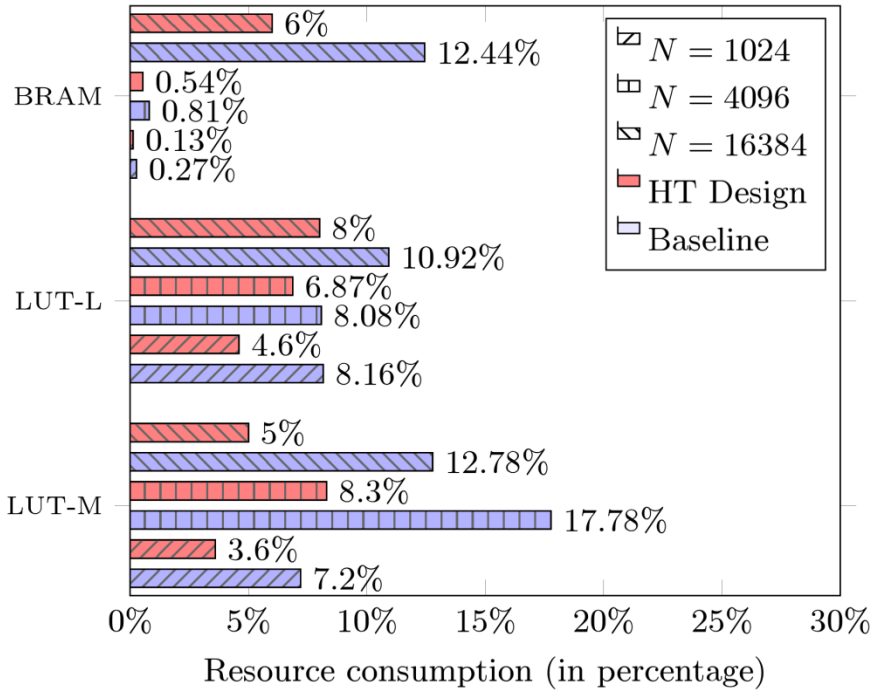
- Resource consumption of the proposed *SPN*
  - Baseline: HT Design implemented without applying in-place permutation
  - Amount of LUT-L: mainly determined by data parallelism
  - Amount of BRAM18: reduced by 50% through in-place permutation in time
  - Scalable wrt. problem size and data parallelism

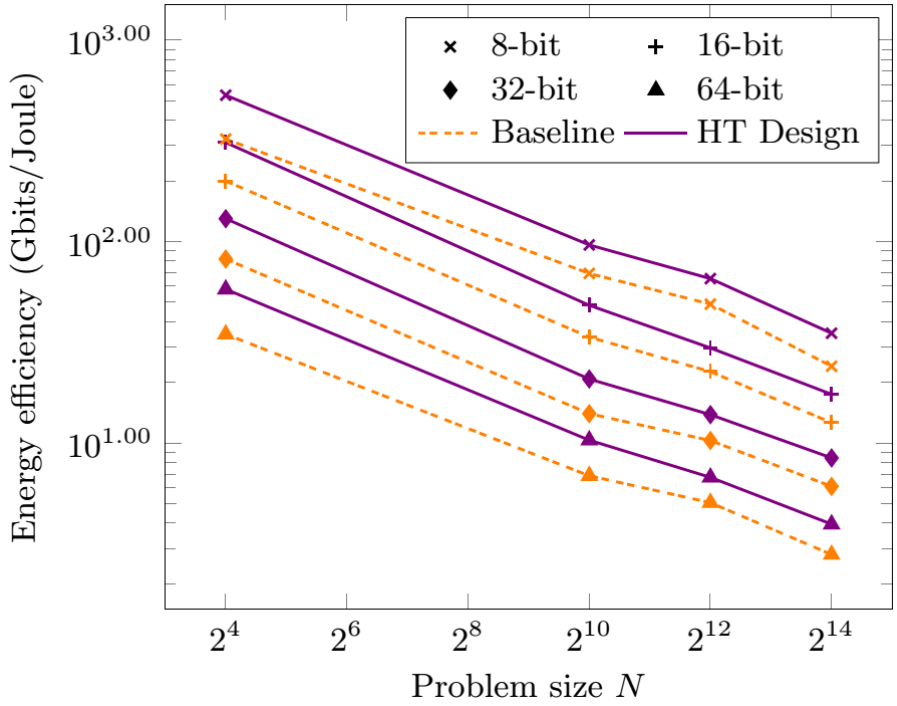




Experimental Results (2)

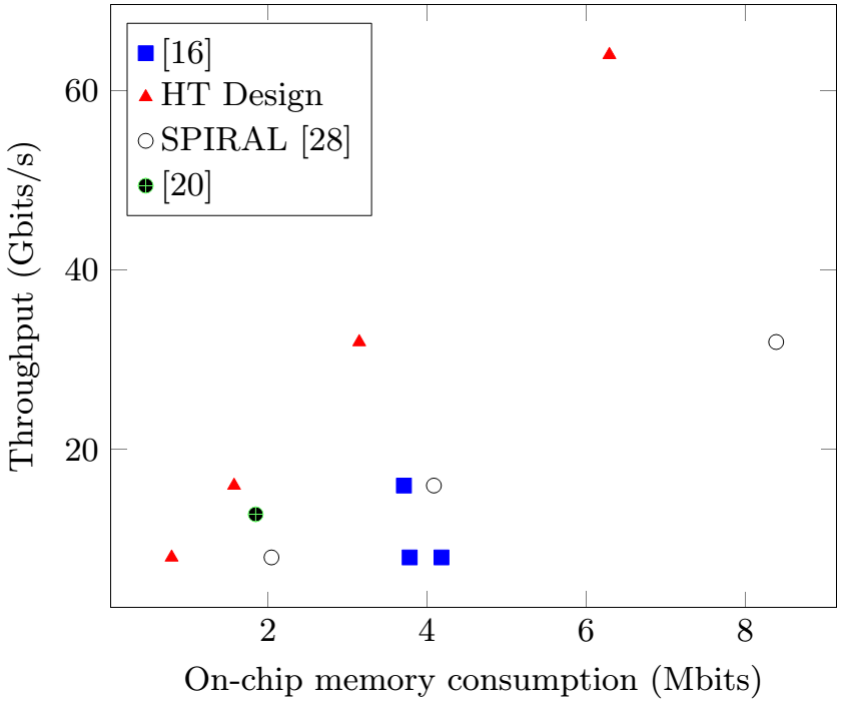
- Performance of the proposed designs for sorting
  - Utilizations of BRAM, LUT-L, LUT-M are reduced significantly
  - 33%~67% energy efficiency improvement compared with the baseline





Experimental Results (3)

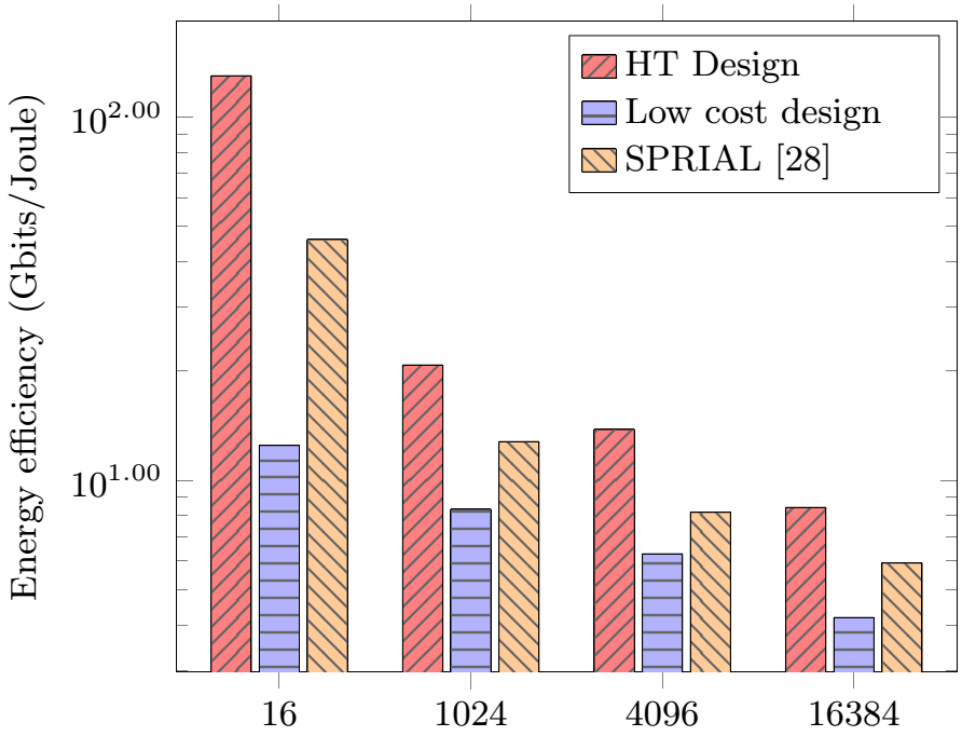
- Memory efficiency comparison
  - [16] (FPGA '11): 43K-key or 21.5K-key data sequences
  - [20] (VLDB '12): for sorting data sets consisting of 8-key data sequences
  - [28] (DAC '12): 16K-key streaming data sequences
  - Our designs achieve
    - 2.3x~5.3x better memory efficiency than [16]
    - 1.5x~2.6x better memory efficiency than [28]
  - All our design points are dominating



Memory efficiency comparison of various design

Experimental Results (4)

- Energy efficiency comparison
  - Data parallelism: 4
  - All pipelined to achieve 250 MHz
  - Achieves the same throughput
  - Problem size: 16~16384
  - [28]: a hardware generator for streaming sort
  - Up to 60% energy efficiency improvement



Energy efficiency comparison of various designs

Conclusion and Future Work

- Conclusion
  - Streaming sorting architecture
    - Scalable with data parallelism and problem size
    - Efficient inter-stage communication realization
    - “Programmable” streaming permutation network
    - Optimal memory efficiency
    - Demonstrates trade offs between throughput, area and latency
- Future work
  - Design framework for automatic application-specific energy efficiency optimizations on FPGA

Thanks!

Questions?

Ganges.usc.edu/wiki/TAPAS