High Throughput Large Scale Sorting on a CPU-FPGA Heterogeneous Platform

Chi Zhang, Ren Chen, Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California

May 23, 2016

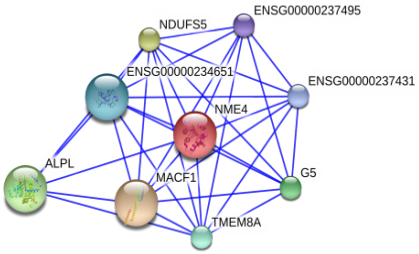
Outline

- Introduction
- Background and Related Work
- Hybrid Mapping on a CPU-FPGA Platform
- Experimental Results
- Conclusion and Future Work

Applications involving Large Scale Sorting

Online social networks

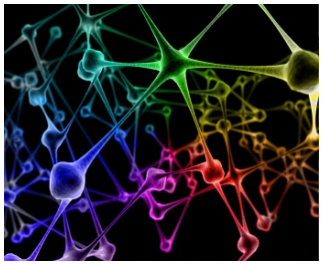
Protein interactions



WWW

Neural network

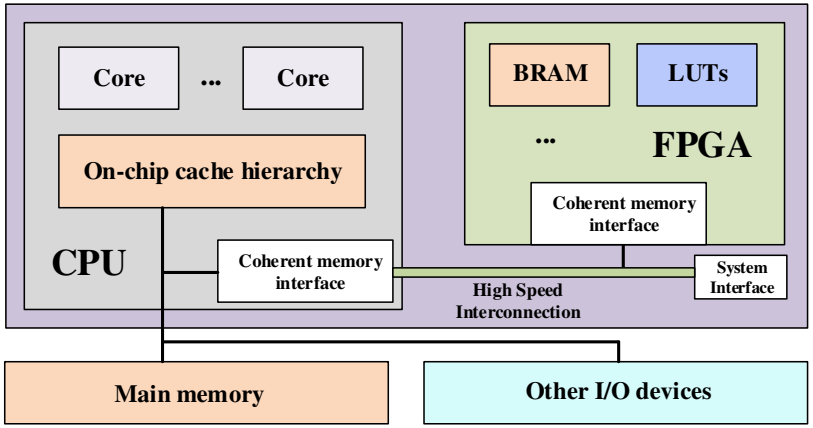
Air traffic network



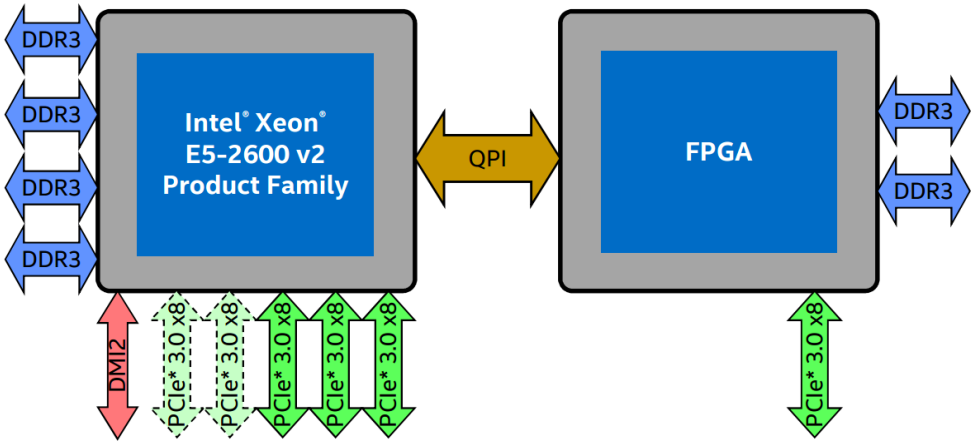
Citation networks

CPU-FPGA Heterogeneous Platform

- FPGA and CPU share the coherent cache system
- FPGA implements its own on-chip cache for fast data access
- FPGA can access data through high speed FPGA-cache interconnection
- FPGA has the same virtual address space with the host CPU application



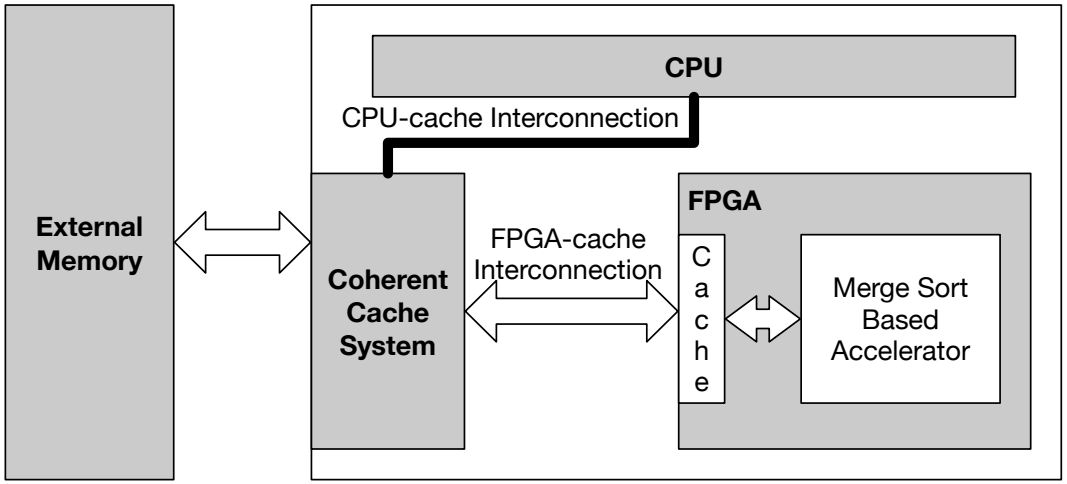
CPU-FPGA shared memory system



Intel QuickAssist QPI FPGA Platform [1]

Problem Definition (1)

- Input: unsorted *N*-key data array in external memory (32-bit keys)
- Memory system: a shared coherent cache system + external memory
- Output: sorted *N*-key data array in the external memory
- FPGA data access: FPGA on-chip cache with cache line granularity



Target CPU-FPGA heterogeneous architecture

Problem Definition (2)

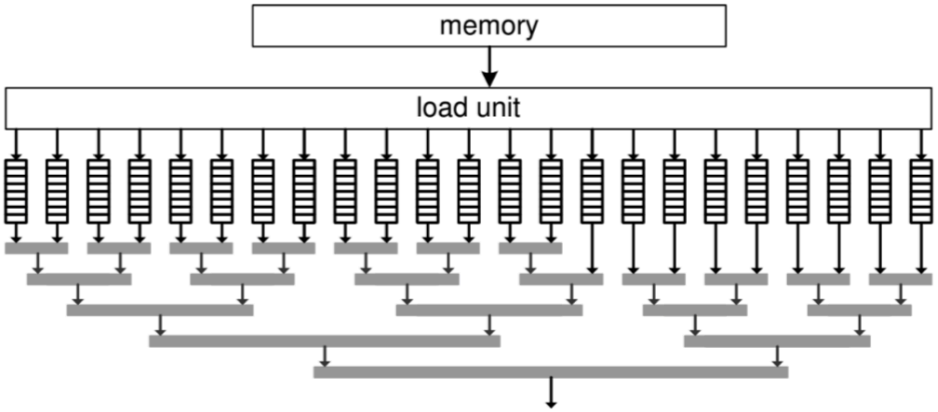
- Performance Metric
  - Throughput
    - Defined as the number of data items sorted per second (Gbytes/s)

Outline

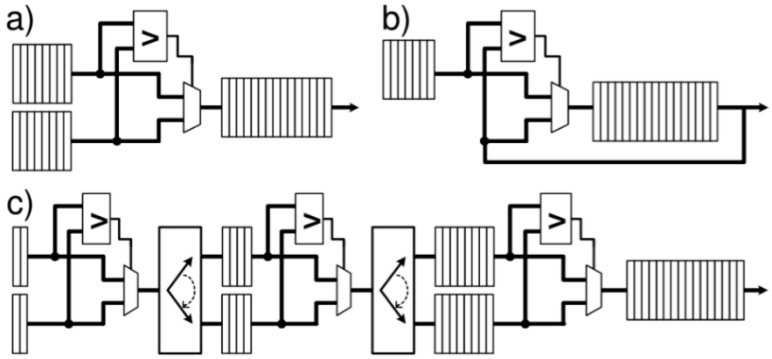
- Introduction
- Background and Related Work
- Hybrid Mapping on a CPU-FPGA Platform
- Experimental Results
- Conclusion and Future Work

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

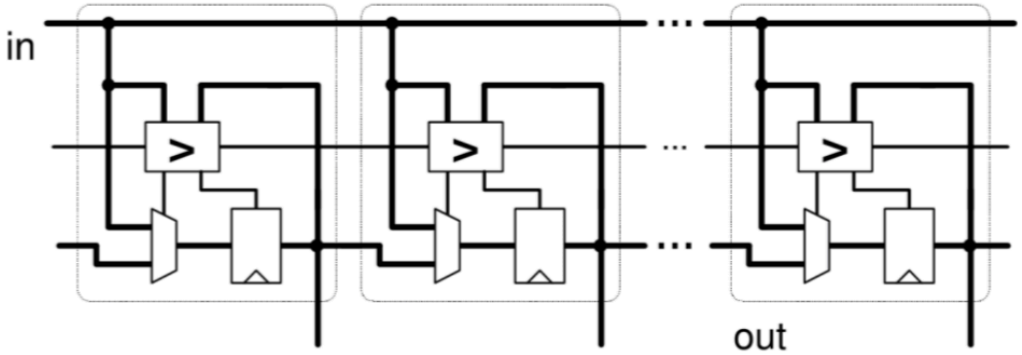
- Tree merge sorter
- FIFO-based merge sorter
- Insertion sorter
- FIFO & Tree
- Partial run-time reconfiguration
- **2 GB/s** using on-chip memory



Tree merge sorter



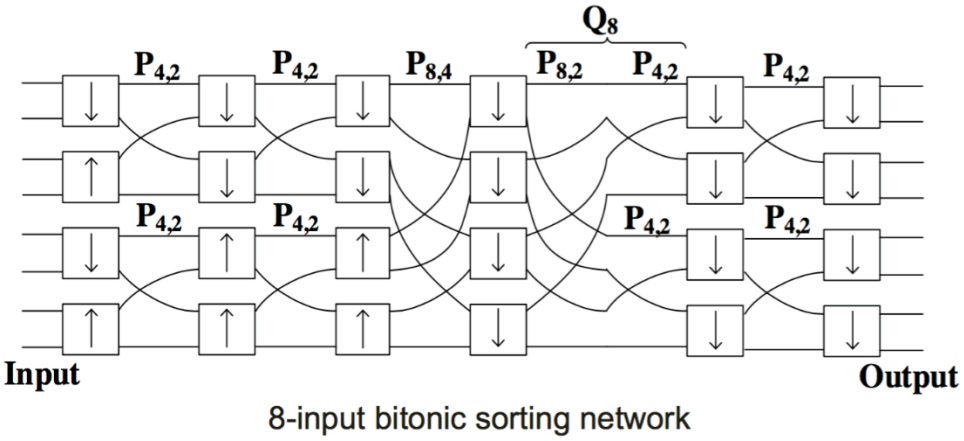
FIFO-based merge sorter



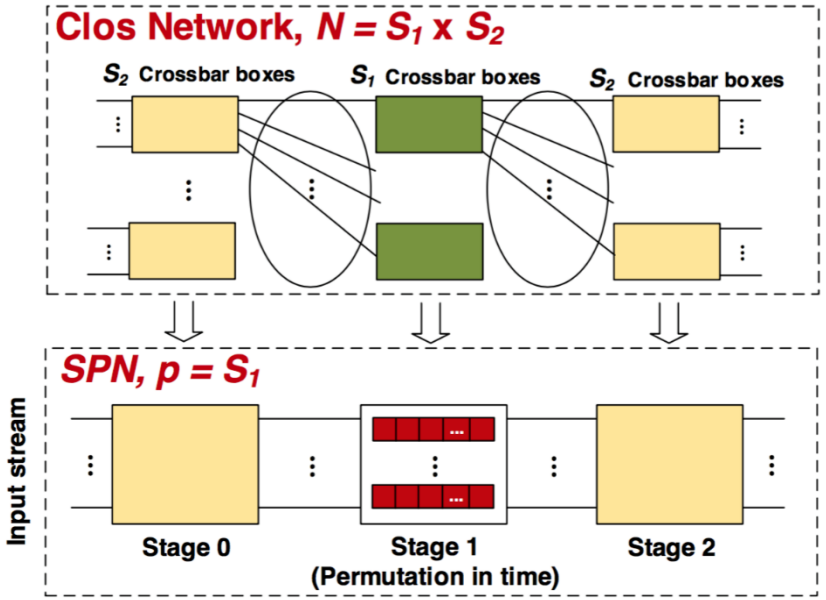
Insertion sorter

Related Work (FPGA '15, Ren Chen)

- Bitonic sorting network based
- "Folded" Clos network to perform inter-stage communication
- Support continuous data streams to maximize throughput
- Highly-optimized for memory and energy efficiency



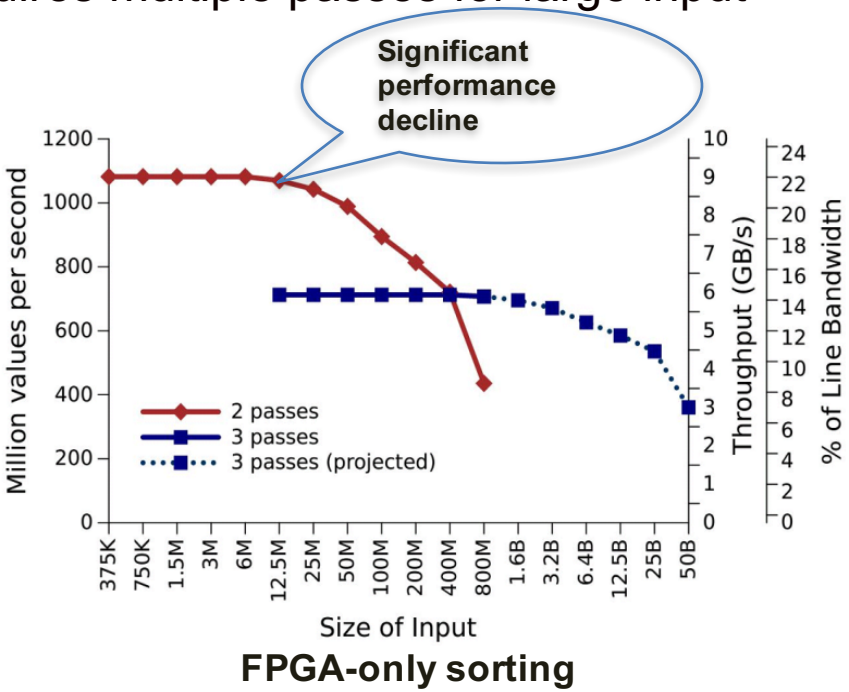
8-input bitonic sorting network



Related Work

- Drawbacks of the state-of-art design on FPGA-only
  - High throughput guaranteed only using on-chip memory
  - Sorting can't be fully pipelined and requires multiple passes for large input size

- We propose a hybrid mapping on a CPU-FPGA heterogeneous platform
  - FPGA accelerates sorting each sub-block
  - CPU merges sub-blocks in serial
  - Concurrent processing on FPGA and CPU



FPGA-only sorting

Outline

- Introduction
- Background and Related Work
- Hybrid Mapping on a CPU-FPGA Platform
- Experimental Results
- Conclusion and Future Work

Sorting on any Platform: Memory?

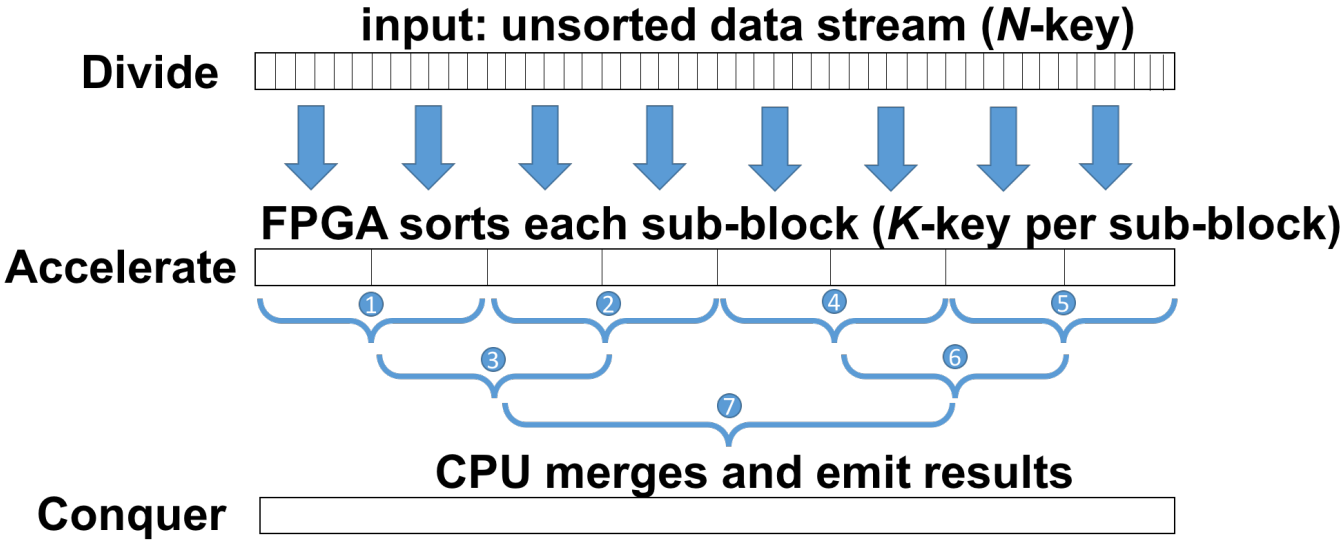
- **Theorem [..80's]**: To sort *N*-key sequence on any hardware architecture using a single pass,

  $\Omega(N)$ on-chip memory is needed.

  - Provides an upper bound for the size of input sequence that a FPGA can sort using a single pass.
  - In order to reduce external memory I/O operations, we propose a hybrid mapping on a shared memory CPU-FPGA platform.

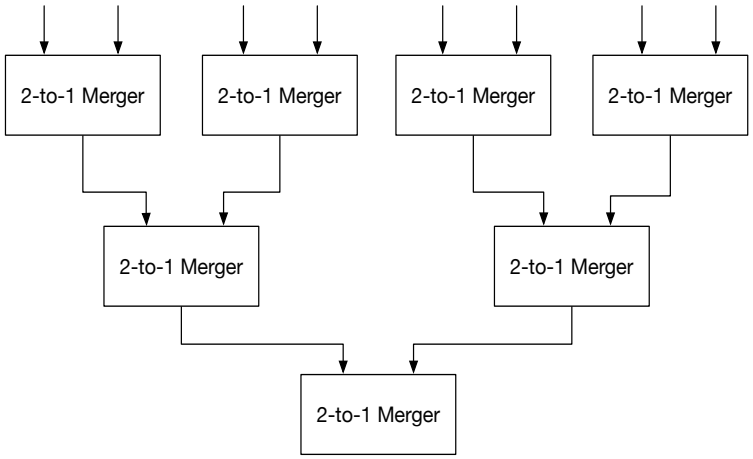
Hybrid Mapping on a CPU-FPGA Platform

- Divide *N*-key unsorted input data stream into sub-blocks, each contains *K* keys
- Use FPGA sorting accelerator to sort each sub-block in a single pass (sub-problem fits in on-chip memory)
- Use CPU to merge sub-blocks **concurrently** into a sorted sequence

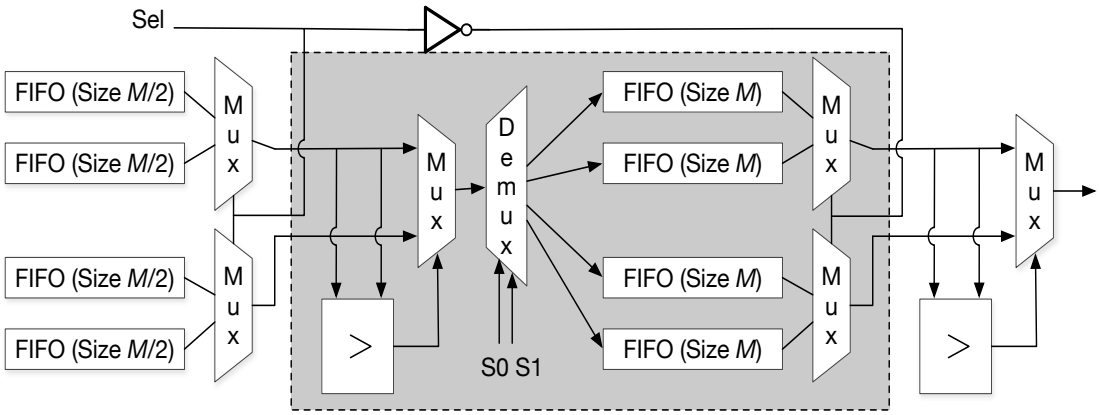


Merge Sort Accelerator (MSA) (1)

- Merge sort based – reduced hardware complexity
- Low data parallelism for merge sort tree at the root
- FIFO-based merger: 2-to-1 mergers in each stage, reuse input FIFOs
- Merge Sort Accelerator: pipelined FIFO-based merger



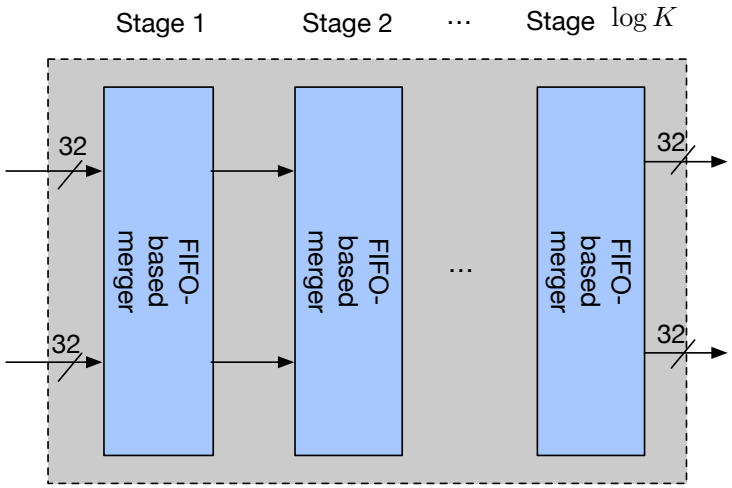
Merge sort tree (depth=3)



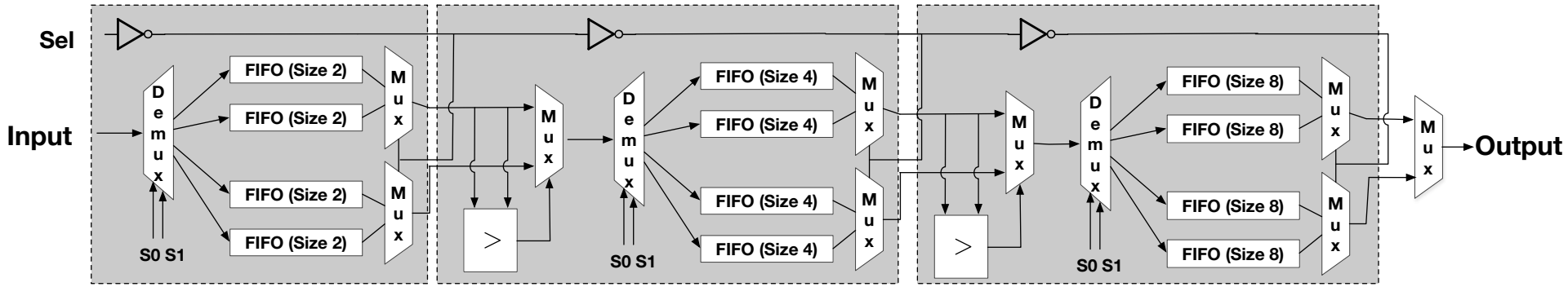
FIFO-based merger

Merge Sort Accelerator (MSA) (2)

- Input: Stream of input data
- Output: Block of *K* keys in sorted order
- Sorts *K*-key data sequence in $\log K$ stages
- Overlaps two sorts of size *K*
- Throughput: One key/cycle
- Latency: $2 + 4 + \cdots + K = O\left(2^{\log K}\right) = O(K)$
- Memory consumption: $4(2 + 4 + \cdots + K) = 4K + o(.)$



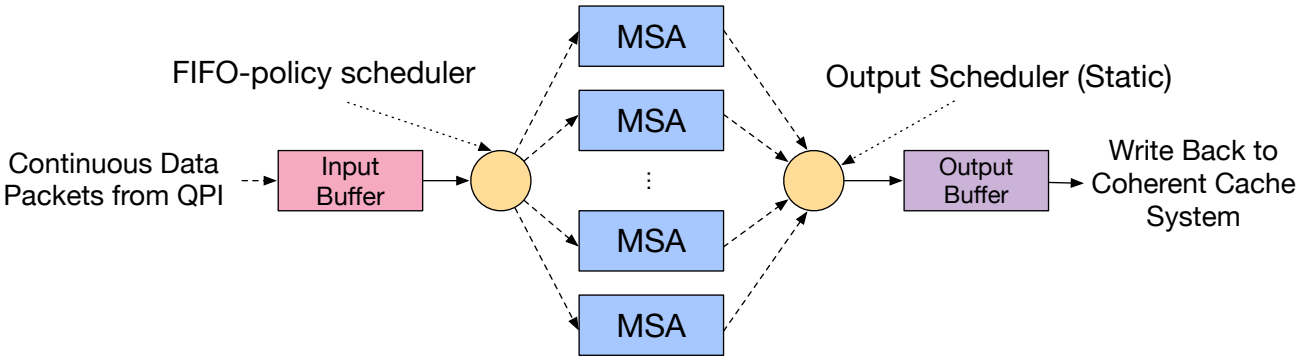
Merge Sort Accelerator



3-stage MSA (*K* = 8)

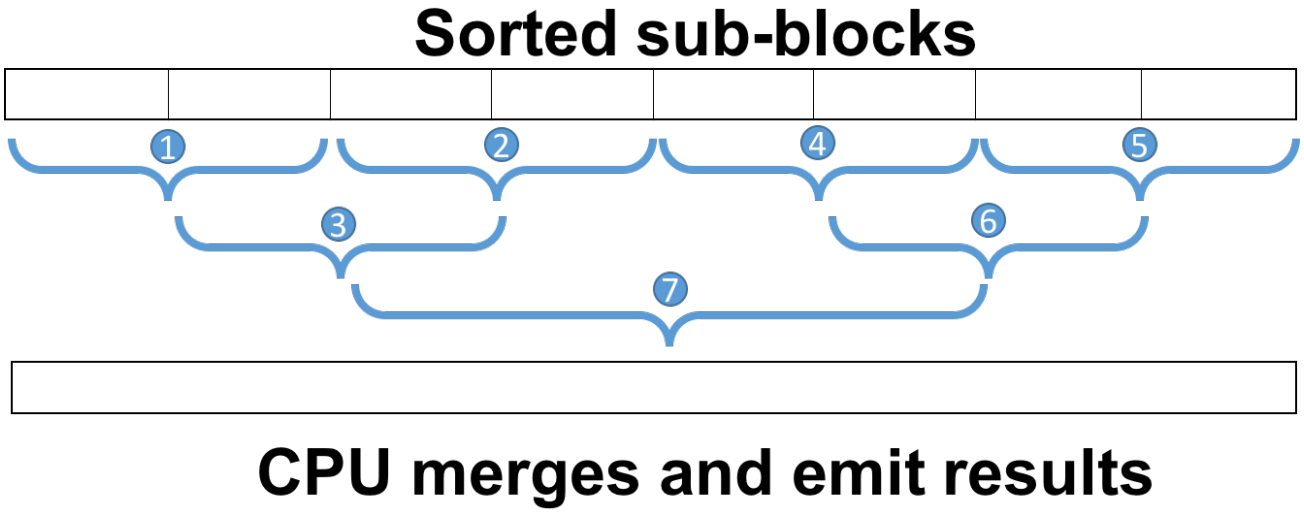
Accelerator Function Unit (FPGA)

- Input: Continuous data packets from FPGA on-chip cache
- Buffer management:
  - Input: FIFO-policy, Output: static
- Task parallelism p: # of Merge Sort Accelerator (MSA) in parallel
- Resource management tradeoff
  - Memory consumption: $(4pK)$
  - # of stages in each MSA vs. # of MSAs in parallel (task parallelism)



Serial Merge (CPU)

- Time complexity: $O(N \log \frac{N}{K})$
- Merge order: **iteratively** from left to right
- Support **concurrent processing** with FPGA
- Synchronization with FPGA: shared flag in coherent cache system



Design Tradeoff Analysis

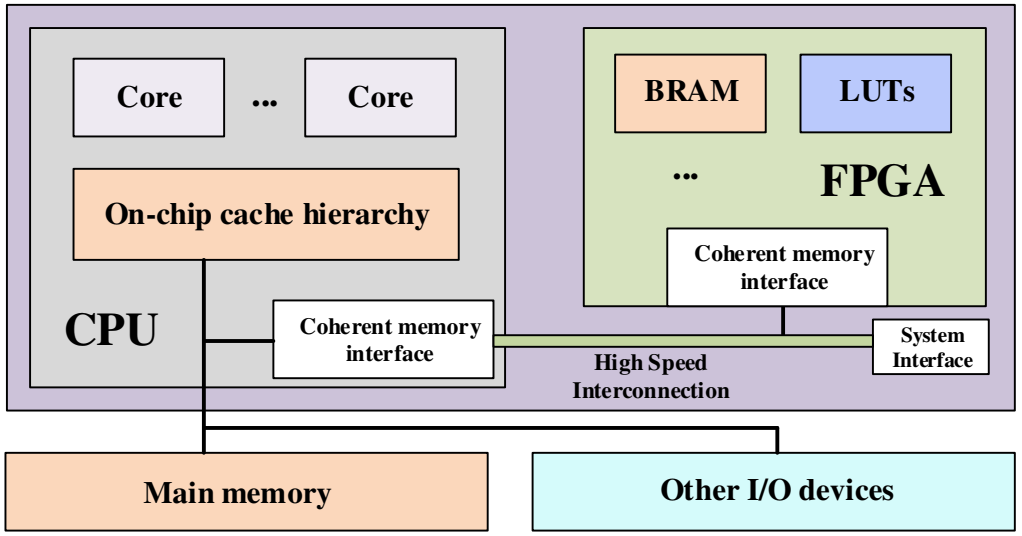
- Given *N*, on-chip memory size, memory bandwidth,
  Choose *p* and *K*
- CPU serial merge time complexity: $O\left(N \log \frac{N}{K}\right)$, where *N* is the input size and *K* is the sub-block size.

- FPGA resource management tradeoff
  - FPGA on-chip memory consumption: $(4pK)$
  - # of stages in each MSA vs. # of MSAs in parallel (task parallelism *p*)

- Throughput
  - Given FPGA-cache interconnection bandwidth and on-chip memory (= product of p and K), there exists a pair (*p, K*) to maximize the throughput

Optimization

- Optimize for FPGA on-chip cache
  - Sub-block fits in FPGA on-chip cache
  - High cache hit rate due to spatial locality
- Optimize for memory access
  - FPGA cache pre-fetching



CPU-FPGA shared memory system

Outline

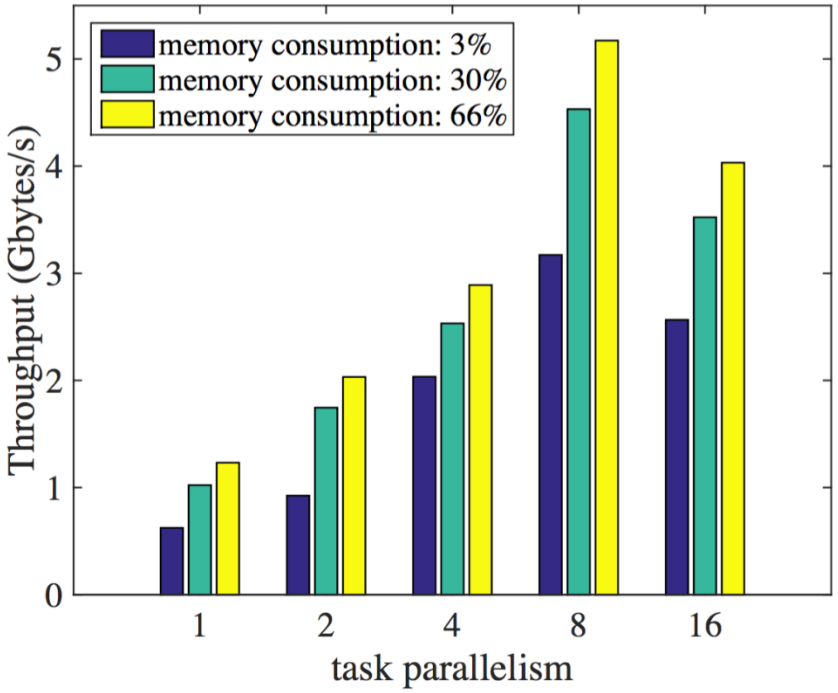
- Introduction
- Background and Related Work
- Hybrid Mapping on a CPU-FPGA Platform
- Experimental Results
- Conclusion and Future Work

Experimental Setup

- Platform and Tools
  - Intel QuickAssist FPGA QPI Platform
  - Altera Stratix V FPGA + Intel Xeon E5-2600 v2 processors (2.8 GHz)
  - **6 Gbytes/s** [1] FPGA-cache interconnection bandwidth (Intel QPI)
  - 128 Kbytes 2-way set-associative FPGA on-chip cache
- Input data sequences in external memory
  - Random data (32-bit keys)
- Performance metric
  - Throughput

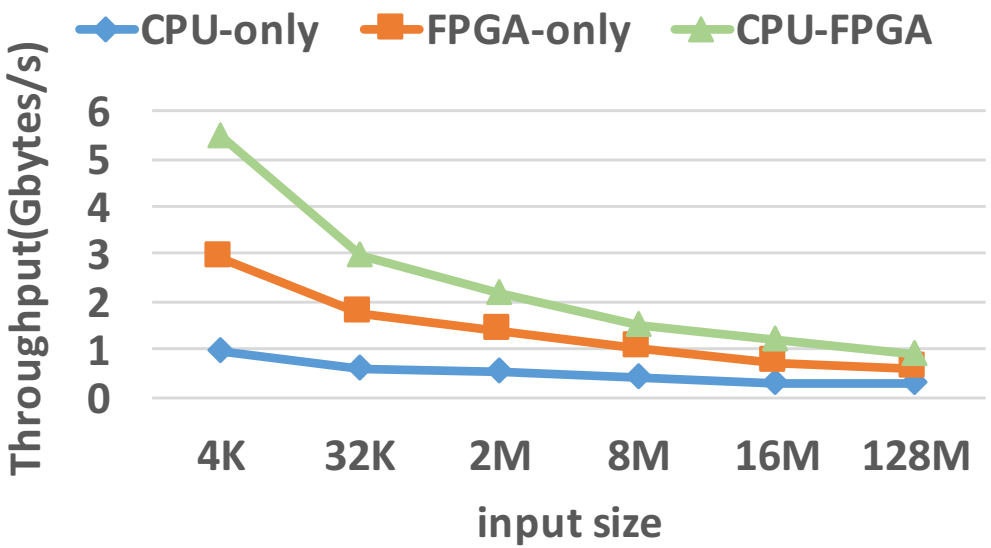
Experimental Results (1)

- Design tradeoff exploration
  - Observation: FPGA data consumption rate matches FPGA-cache interconnection bandwidth: $4pf = B$
  - Altera FPGA $f = 200\ MHz$
  - Intel QPI $\boldsymbol{B = 6\ Gbytes/s}$
  - **For each *p* vary *K***
  - (Using available on-chip memory) maximum throughput achieved when task parallelism is 8
  - Input size: 4K-key sequence
  - Experimental results match observation
  - Best configuration: $p = 8,\ K = 16384$



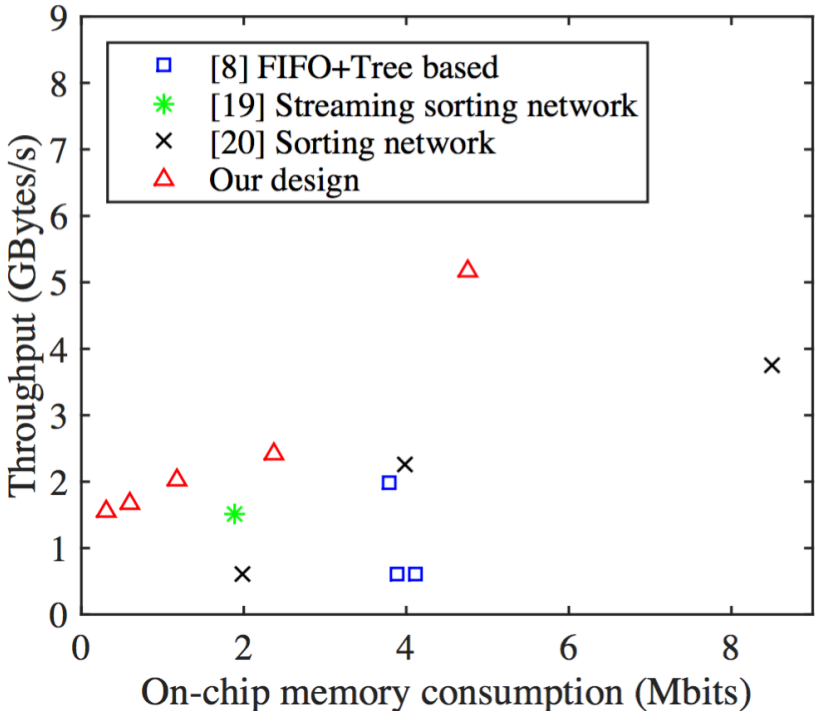
Experimental Results (2)

- Performance compared to CPU-only and FPGA-only baselines
  - Fix $p = 8,\ K = 16384$ for FPGA, vary the input size
  - CPU-only: quicksort in C++ standard library
  - FPGA-only: MSA without help of the CPU, projected for large input size
  - Problem size:
    4K-keys ~ 128M-keys
  - 2.1x ~ 2.5x compared to FPGA-only baseline
  - 2.1x ~ 5.3x compared to CPU-only baseline



Experimental Results (3)

- Performance compared with the state-of-art
  - [8] (FPGA '11): 43 K-key or 21.5 K-key data sequences
  - [19] (VLDB '12): for sorting data sets consisting of 8K-key data sequences
  - [20] (DAC '12): 16 K-key streaming data sequences
  - Our design: 4K-key data sequences in external memory
  - 2.59x ~ 3.90x compared to [8]
  - 1.38x ~ 3.88x compared to [20]



Outline

- Introduction
- Background and Related Work
- Hybrid Mapping on a CPU-FPGA Platform
- Experimental Results
- Conclusion and Future Work

Conclusion and Future Work

- Conclusion
  - Accelerated sorting on CPU-FPGA platform
    - Merge Sort Accelerator in FPGA
    - Serial Merge in CPU

- Future Work
  - Accelerate FPGA data access by memory pipelining and pre-caching

Thanks!

fgpa.usc.edu

Questions?