Inadequacy of Processor Load-Store Queues

- Conventional LSQ allocation policy is not suitable for spatial computing
  - In a processor, decoding conveys the correct sequential order of requests at the memory interface
  - Spatial circuits have no equivalent phase to Decode

Overcoming the Limitations of Static Scheduling

- Static scheduling (HLS tools)
  - Inferior schedules when memory accesses cannot be disambiguated at compile time
- Dynamic scheduling
  - Analysis of data dependencies at runtime
  - Stalling only if a data or a control hazard occurs
  - Requires a memory interface that supports out-of-order execution

Resource utilization and execution time of the dynamically scheduled designs using our LSQ, normalized to the corresponding static designs produced by Vivado HLS.

Static scheduling (HLS tool):
- Inferior schedules when memory accesses cannot be disambiguated at compile time
- Requires a memory interface that supports out-of-order execution

Dynamic scheduling
- Allocate entries for all the memory operations of a group at once
- The arguments of the memory accesses can arrive out-of-order

Conventional LSQ allocation policy is not suitable for spatial computing
- Allocate entries for all the memory operations of a group at once
- The arguments of the memory accesses can arrive out-of-order

Supplying Sequential Order to the LSQ

Basic idea: allocate groups of memory accesses depending on the control flow of the application

Groups
- Sequences of accesses which are statically predefined
- If one access of a group executes, all other accesses belonging to the same group will eventually execute
- The notion of basic blocks already exists in HLS tools

An Out-of-Order Load-Store Queue for Spatial Computing

- Linked entries to the access ports
- The notion of basic blocks already exists in HLS tools

Experimental Results

- The high sensitivity to the number of queue entries is in line with what others achieved in conventional LSQ designs
- Changing the number of groups and the way the ports are organized has minimal influence on cycle time

Significant improvement in execution time (up to 3x) in all benchmarks
- The LSQ designs dynamically resolve dependencies and increase throughput, with usually an acceptable impact on clock period (CP)