Paper ID: 83

# Array Partitioning Method for Streaming Dataflow Optimization in High-level Synthesis

Renjing Hou
Beijing University of Posts and Telecommunications
hourj@bupt.edu.cn

# Outline

- Introduction
- Preliminaries
- Method
- Experiments
- Conclusions

# Introduction

- High-level Synthesis
  - ➢ reduce the development time
  - ➢ improve the testing and simulation ability
  - ➢ narrow the gap between software engineers and FPGA

  - – Input: behavior-level languages (e.g.,C/C++/OpenCL)
  - – Output: RTL descriptions(e.g., Verilog/VHDL)
  - – Use synthesizable control knobs to generate good-quality RTL designs
    - – Pragmas: pipeline, loop unroll, dataflow, array partition, etc.

# Introduction

- Dataflow
  - one of the important optimization methods to achieve task-level parallelism
  - Two micro-architectures: the ping-pong dataflow and the streaming dataflow
  - Streaming Dataflow: Easy to implement, higher accessing efficiency , but with strict constraints

- Research motivation
  - To broaden the usage scenarios of streaming dataflow

# Introduction

- Related works
  - Dataflow:
    - Design Space Exploration(DSE), to find suitable parameter values
      - Hi-ClockFlow
      - DataMaster
  - Array partitioning:
    - focus on standalone scenarios, to improves hardware efficiency
  - no research has combined them together
    use partition method to improve streaming dataflow

# Preliminaries

- Dataflow
  - Improve the parallelism between coarse grained tasks
  - categorized into two micro-architecture according to different memory channel implementations

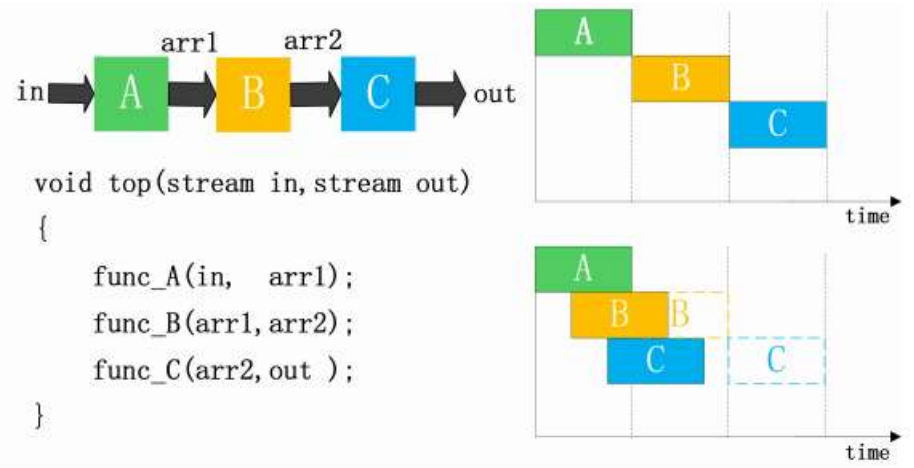


Fig. 1 Dataflow can achieve task-level parallelism.

- Ping-pong Dataflow
  - Two buffers take turns reading and writing

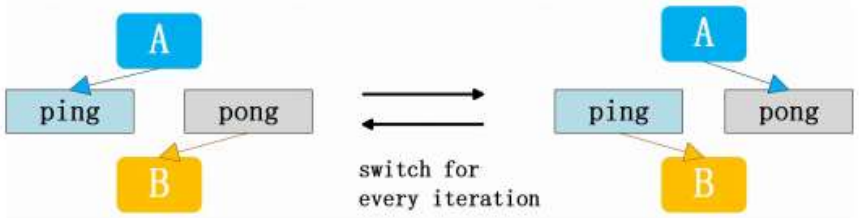


Fig. 2 The illustration of ping-pong buffers.

- Streaming Dataflow
  - First in first out(FIFO) channel

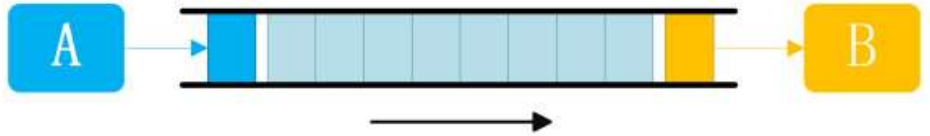


Fig. 3 The illustration of streaming buffers.

# Preliminaries

- Streaming dataflow(VS ping-pong dataflow)
  - Easy to implement
  - higher accessing efficiency
  - strict constraints: read and write operations must be consistent and serial in order

# Preliminaries

- Array partition
  - Implementing parallelism on memory access
  - Multiple partitioning strategies

    block、cyclic、complete

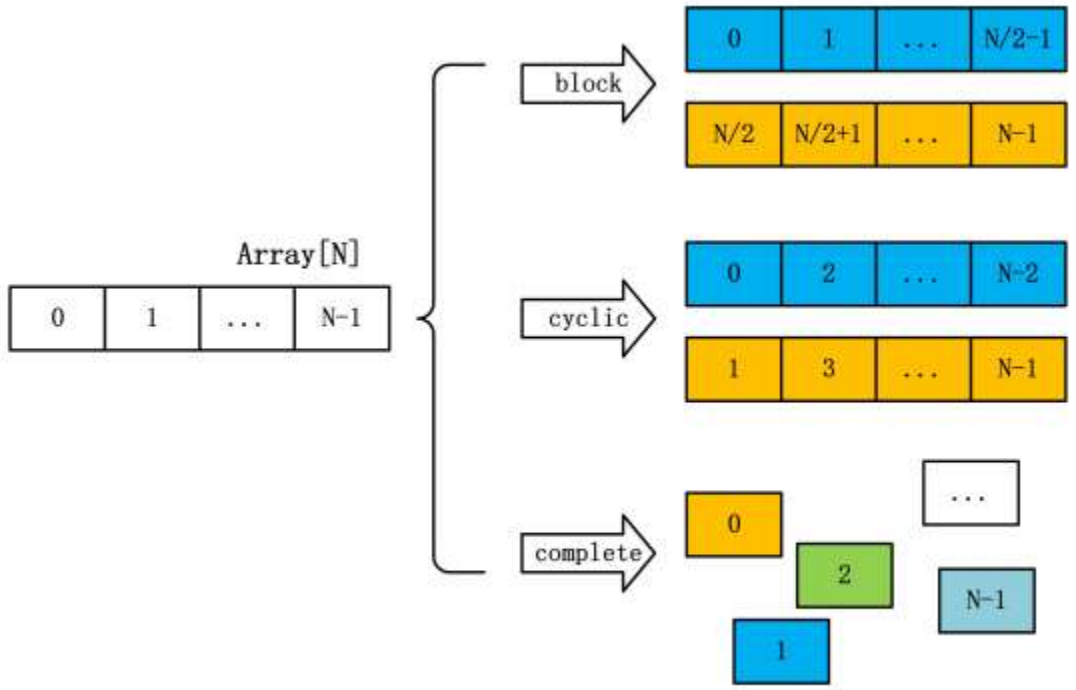


Fig. 4 Three strategies of *array_partition* in HLS

# Preliminaries

- Streaming dataflow: better preference & strict constrains
- HLS tools use ping-pong buffer when it cannot determine whether memory access is performed serially.
- By converting the non-serial access to continuous access in some way, then we can use streaming dataflow to further improve performance

- Problem formulation

Given an array $T$ with $n$ elements, each element $T[i]$ has its own read/write access $Ac[i]$, where $Ac$ is a permutation from 1 to $n$. The objective is to partition $T$ into $m$ sub-arrays $\{T_1, T_2, ..., T_m\}$, and guarantee that: for each sub-array $T_x$ $(x \in \{1, ..., m\})$, $Ac[k+1] = Ac[k] + 1$ $(k \in \{1, ..., size(T_x) - 1\})$.

# Method

- Overall flow

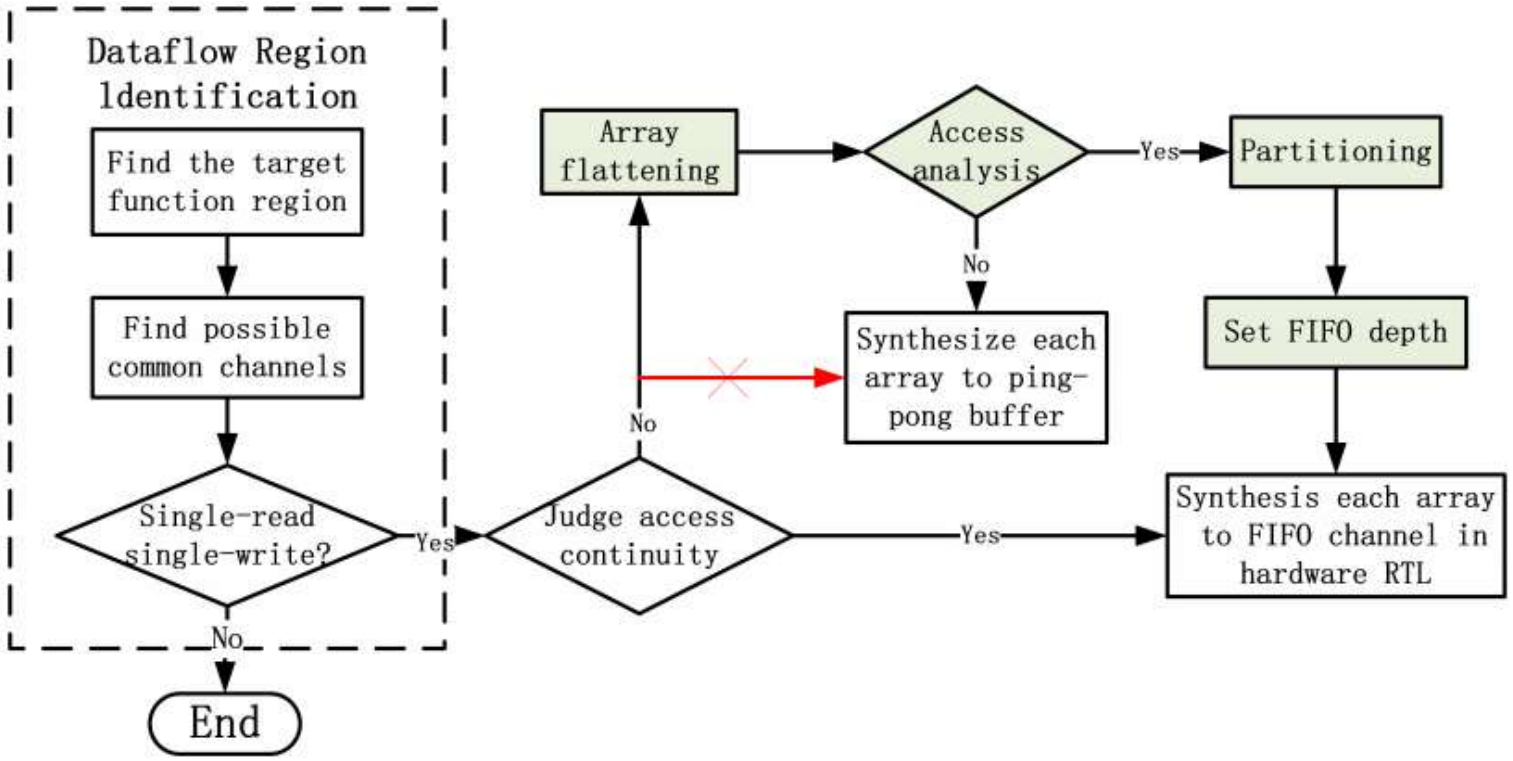


Fig. 5 The overall flow of the proposed method.

# Method

- Access Analysis & Partitioning

$$
\begin{aligned}
&C(n,t,m) = B(1,t,m), B(2,t,m), ..., B(n,t,m)\\
&B(i,t,m) = A_1(i,m), A_2(i,m), ..., A_t(i,m), i \in [1,n]\\
&A_j(i,m) = p_j^i, p_j^i + 1, ..., p_j^i + m - 1, j \in [1,t]\\
&p_j^i = (k-1) * n * m + m * (j-1).
\end{aligned} \quad (1)
$$

$$
S_j(n,m) = A_j(1,m), A_j(2,m), ..., A_j(n,m), j \in [1,t] \quad (2)
$$

- n:cycle number
- t:block number
- m:block length

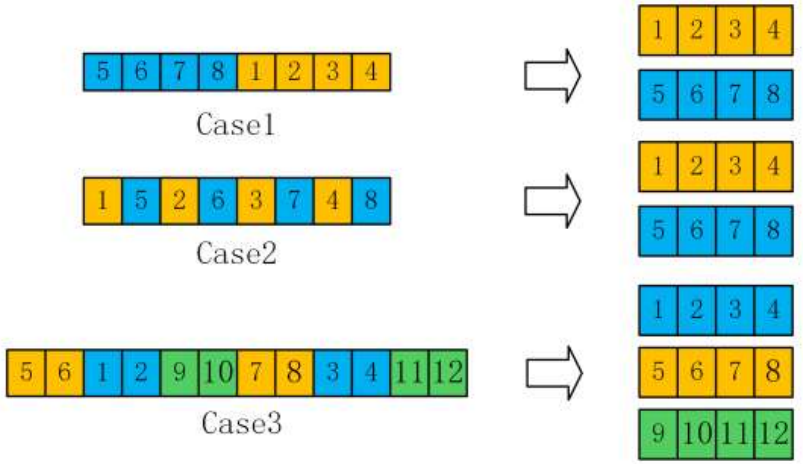


Fig. 6 Three example cases to show the key strategy.

TABLE I Formula values of the cases

| Case | C | B | $A_j$ | $S_j$ |
|---|---|---|---|---|
| Case-1 | $(1,2,4)$ | $(1,2,4)$ | $(1,4)$ | $(1,4)$ |
| Case-2 | $(4,2,1)$ | $(i,2,1)$ | $(i,1)$ | $(4,1)$ |
| Case-3 | $(2,3,2)$ | $(i,3,2)$ | $(i,2)$ | $(2,2)$ |

# Experiments

- Environment
  - Xilinx Vitis HLS (version 21.2)
  - CentOS
  - 48 Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz
  - experiment results are obtained via synthesis and routing with Xilinx Vivado 21.2

- Case Study
  - Convolutional Neural Network
  - Use the method for the part of the convolutional layer to the pooling layer

# Experiments

- Access Analysis & Partitioning

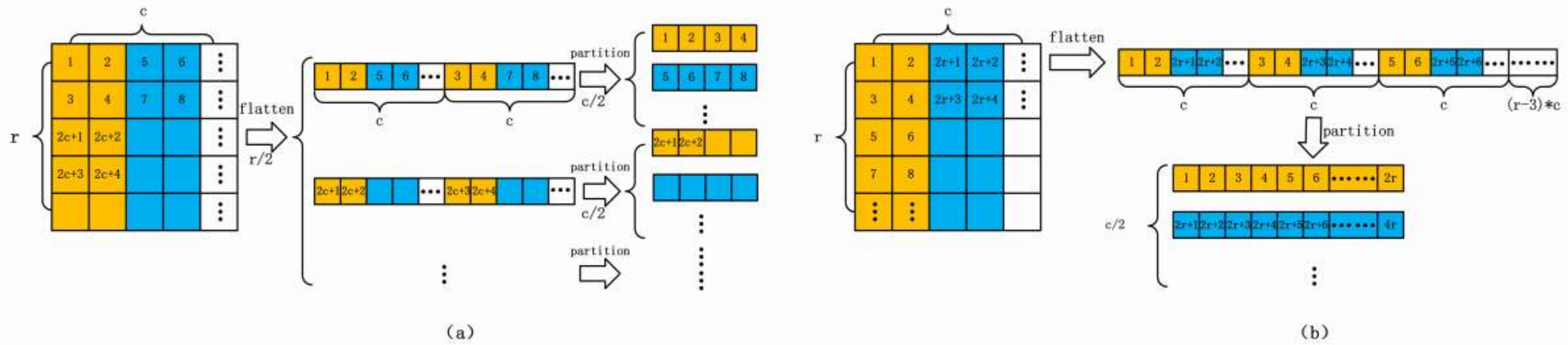


Fig. 7 The illustration of the CNN case study: (a) is the row-major access between blocks, and (b) is the column-major access.

# Experiments

- Results
  - The latency (i.e., clock cycle * timing) of streaming dataflow is 28.6% less than the latency of ping-pong dataflow
  - Due to the impact of the memory bank switch and the increasing number of FIFOs caused by array partitioning, the power of the streaming dataflow is 7.2% higher than the power of the ping-pong dataflow

TABLE II QoR of Streaming and Ping-pong Dataflow

| Dataflow | Power (W) | Clock Cycle | Timing (ns) | Latency (ns) |
|---|---|---|---|---|
| Ping-pong | 0.290 | 53293 | 9.169 | 488148 |
| Streaming | 0.311 | 32582 | 10.702 | 348692 |

# Conclusions

- Achievement
  - Proposed an efficient method using array partitioning to ease the constraints of the streaming dataflow.

- Future work
  - analyze and handle the inconsistent access order of read and write to fit the streaming dataflow requirement

Paper ID: 83

# Thanks

Renjing Hou
Beijing University of Posts and Telecommunications
hourj@bupt.edu.cn