Towards Buffers as a Scalable Alternative to Registers for Processor-Local Memory

Julius Roob, Anoop Bhagyanath, and Klaus Schneider

March 23, 2023

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Introduction

- motivation: increase processor Instruction Level Parallelism (ILP)

- need: many processing units (PUs)
- but: **more ports** to registers $\Rightarrow$ $\mathbf{O}(\mathbf{n^2})$

- our research: develop better alternative
- Buffered Exposed Datapath (BED) architectures
- goal: $\mathbf{O}(\mathbf{n})$ for every part
- this paper: **local memory / distributed buffers**

RPTU

Contents

Buffered Exposed Datapath (BED) Architectures

Buffer Architectures

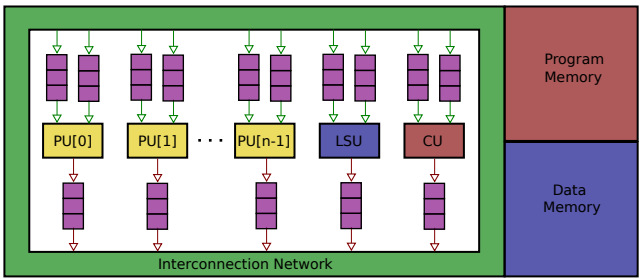
Experiments & Analysis

Conclusions

RPTU

Buffered Exposed Datapath (BED) Architectures

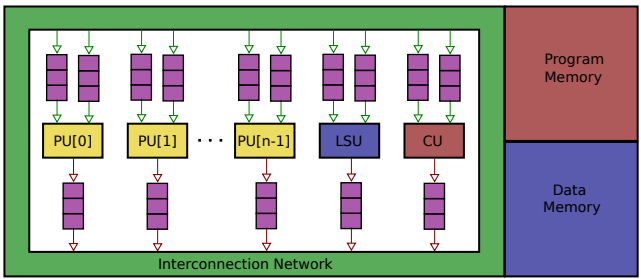
compiler is responsible for data movement through the processor.



- Transport Triggered Arch. (TTA)
- Synchronous Control, Asynchronous Data (SCAD)

Buffered Exposed Datapath (BED) Architectures

compiler is responsible for data movement through the processor.

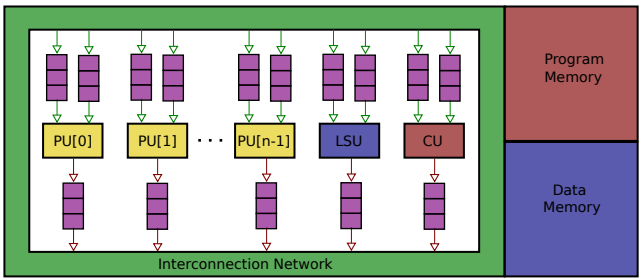



```
...
13: pu0@out -> pu1@in0
14: (lesN, 1) -> pu1@opc
15: $3 -> cu@in1
16: pu1@out -> cu@in0
...
```

RPTU

Buffered Exposed Datapath (BED) Architectures

compiler is responsible for data movement through the processor.



Buffers:

- FIFOs
- Reorder
- Registers

RPTU

Problem Setting (1/2)

presented last year at MBMV:

- virtual FIFOs/channels
- size reduction and bandwidth argument for reorder buffer
- verification

Problem Setting (2/2)

- still building full prototype
- hardware complexity analysis

- ongoing work, this paper is part of the process

- components not all ready for ILP analysis
- but: different buffer architectures have been developed.

Buffer Architectures

following slides will introduce

monolithic:

- register file

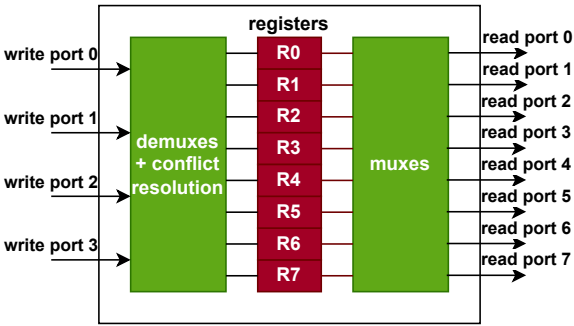
distributed:

- FIFO
- Reorder-Read
- Reorder-Reserve
- Ripple matching

they will then be evaluated in the experiment

RPTU

Central Register File



- many PUs
- accessing the same register
- hardware complexity: **quadratic** for number of ports

First In First Out (FIFO)

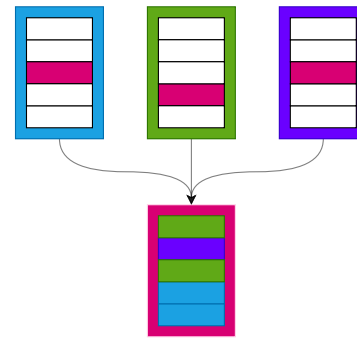
- well-known
- efficient implementations
- pointers or fall-through
- arrival order

Reorder Buffers

- program order of moves
- data from many PUs
- unknown delays

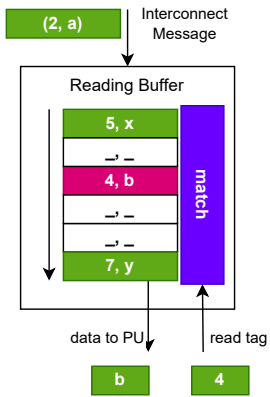
solution: reorder buffers

- will show different variants



RPTU

Reorder-Read



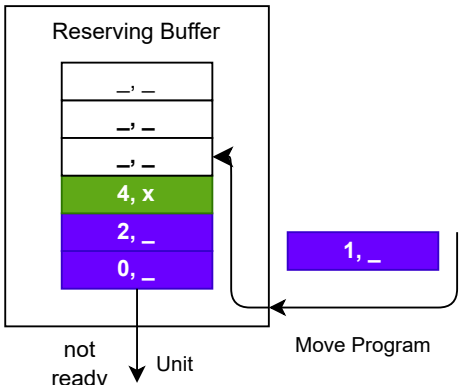
- messages have tags
- program determines order of tags

- incoming messages are stored in order of arrival
- PUs request messages with specific tags
- reordering happens during consumption

(See our MBMV '22 paper and presentation for more details)

RPTU

Reorder-Reserve (1/2)

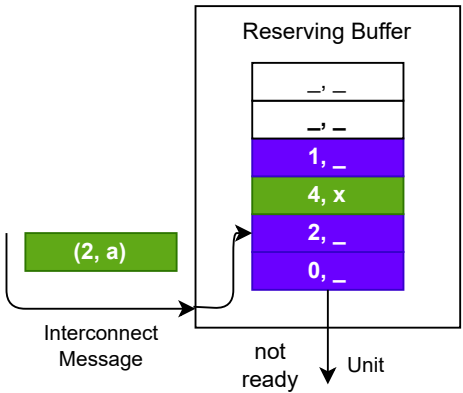


- different approach
- program reserves spaces with tags
- messages are matched against reservations

(SCAD)

RPTU

Reorder-Reserve (2/2)



- different approach
- program reserves spaces with tags
- messages are matched against reservations

(SCAD)

RPTU

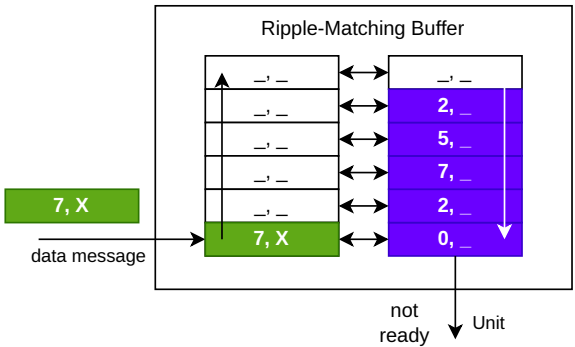
Scheduling Constraints

- latency-dependent: FIFO

program order:

- (mostly) latency-independent: reorder-read
- latency-independent: reorder-reserve

Ripple-Matching (1/3)



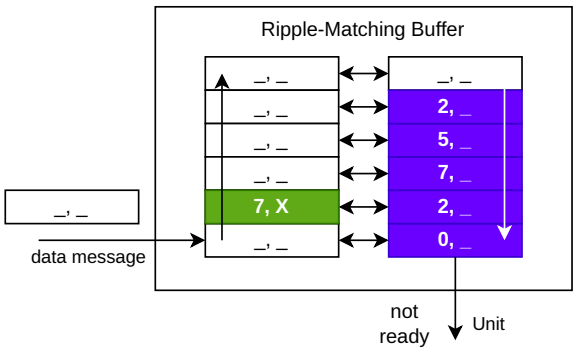
- implements reorder-reserve
- local comparisons

- messages travel along reserved spaces
- stay at first matching place

- starting at head where PU reads
- no delay introduced by ripple

RPTU

Ripple-Matching (2/3)



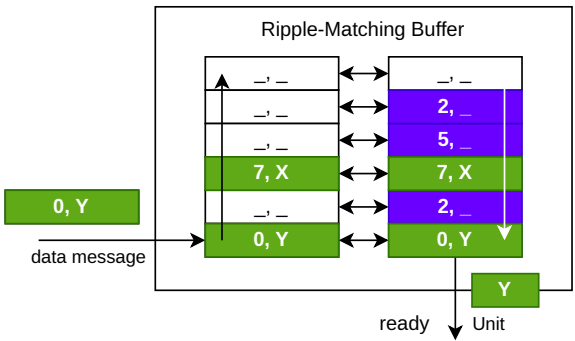
- implements reorder-reserve
- local comparisons

- messages travel along reserved spaces
- stay at first matching place

- starting at head where PU reads
- no delay introduced by ripple

RPTU

Ripple-Matching (3/3)



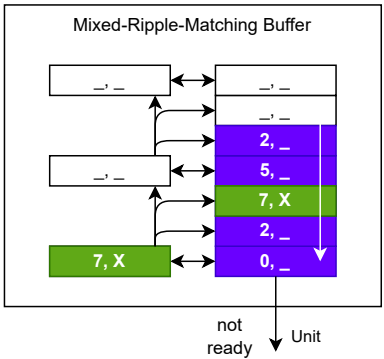
- implements reorder-reserve
- local comparisons

- messages travel along reserved spaces
- stay at first matching place

- starting at head where PU reads
- no delay introduced by ripple

RPTU

Mixed-Ripple



- concept to adapt ripple-matching to different applications
- full ripple-matching needs more registers
- registers vs. critical path

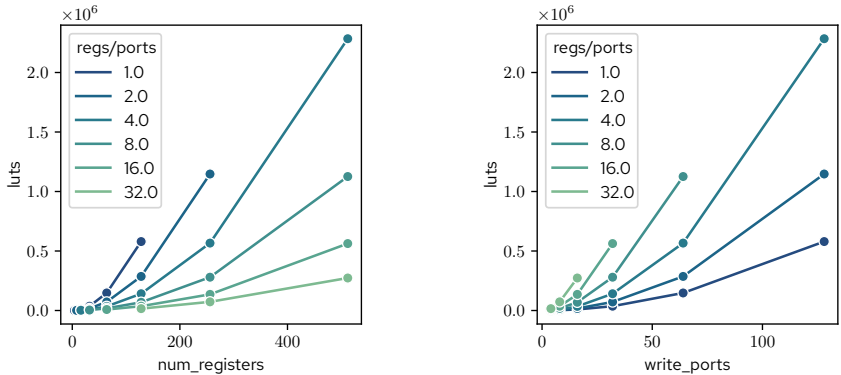
Setup

Experiments

- evaluation of presented architectures
- implement variants for synthesis on Xilinx Ultrascale+
- scaling parameters for registers:
  - number of registers
  - number of read/write ports
- scaling parameters for other buffers:
  - size
  - (tag width)
- metrics: critical path, LUTs and FFs

Resource Usage: Registers

Experiments

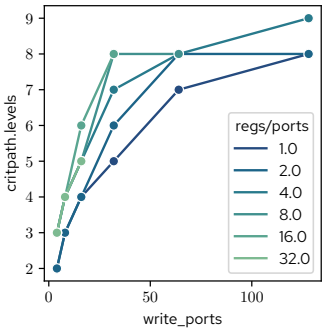


LUT usage of register files with (left) increasing number of registers and (right) increasing number of ports

RPTU

Register Results

Experiments



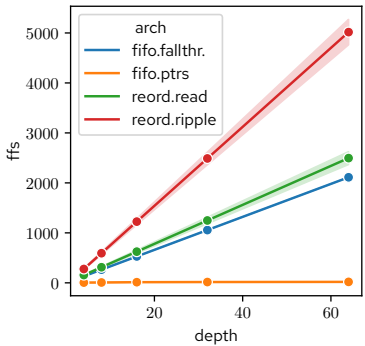
Critical path lengths of registers.

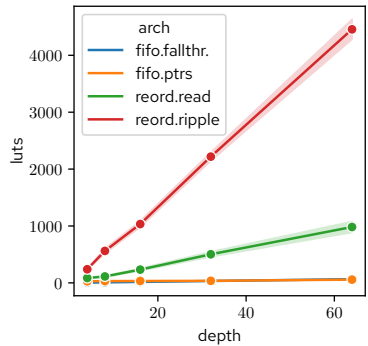
- critical path length looks fine
- but size...
- **quadratic** with number of ports
- => quickly larger than many processors
- arbitration in a single cycle?

RPTU

Resource Usage: Buffers

Experiments



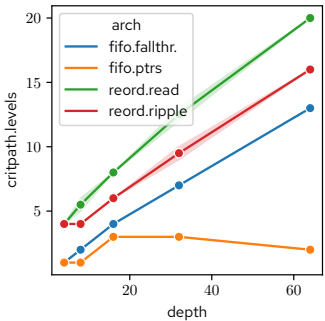


(left) FF and (right) LUT usage of different buffers with growing buffer depth

RPTU

Distributed Buffers

Experiments



Critical path lengths of different buffers.

- pointer FIFO is mapped to BRAM
- ripple-matching still linear
- $\rightarrow$ skid buffers in future work
- all buffers are (quasi-)**linear** size

RPTU

Comparison (1/2)

Experiments

central register file

- **quadratic** growth with port number

distributed buffers

- N buffers for each PU
- **linear** growth

Comparison (2/2)

Experiments

example

- 32 PUs
- 16 intermediate values each

central register file

- 512 registers
- 32 write ports
- 64 read ports
- $\rightarrow$ **562624** LUTs ☹

distributed buffers

- two reorder buffers per PU
- each of depth 8
- reorder-reserve: $338 \times 32 \times 2 = \mathbf{21632}$ LUTs ☺
- reorder-read $124 \times 32 \times 2 = \mathbf{7936}$ LUTs ☺

RPTU

Conclusion

- scalability analysis
- argument for BEDs: buffer scaling with port number **linear**!

Other contributions:

- reorder buffer architectures
- ripple-matching

Questions?

- any questions?

- thank you :)