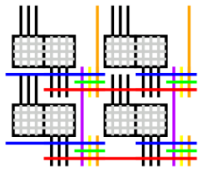
Fast and Flexible FPGA development using Hierarchical Partial Reconfiguration

Dongjoon(DJ) Park, Yuanlong Xiao, André DeHon
<dopark,ylxiao>@seas.upenn.edu, andre@ieee.org



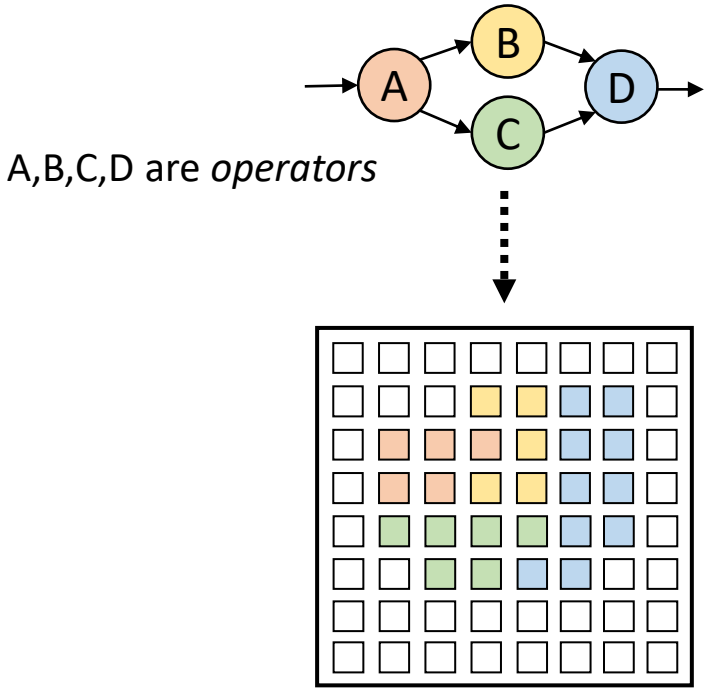
Implementation of Computation Group
University of Pennsylvania

Background

- FPGA *compilation* is slow 😥
  - High-Level-Synthesis
  - Logic Synthesis
  - Placement
  - Routing
  - Bitstream generation
- How slow?
  - CPU, GPU: milliseconds, seconds, minutes?
  - FPGA: minutes, hours, ***days*** ;-(
- Users can’t quickly edit/compile/debug
  - If you imagine a realistic development scenario…

Background

- Why slow?: FPGA's compilation on vendor tool(Intel, AMD) is *monolithic*
  - Compiling/optimizing a large chunk
  - Limited parallelism in Placement/Routing phase



Solution to 🤔 slow compilation?

Background

- Recently, researchers suggested
  Separate Compilations in Parallel with Fixed-sized Pages…

  - Case for Fast FPGA Compilation using Partial Reconfiguration, **FPL 2018**
  - Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks, **FPT 2019**
  - Fast linking of separately compiled FPGA blocks without a NoC, **FPT 2020**
  - Software-like compilation for data center fpga accelerators, **HEART 2021**
  - Rapidstream: Parallel physical implementation of FPGA HLS designs , **FPGA 2022**
  - PLD: Fast FPGA compilation to make reconfigurable acceleration compatible with modern incremental refinement software development, **ASPLOS 2022**
  - HiPR: High-level Partial Reconfiguration for Fast Incremental FPGA Compilation, **FPL 2022**

    …

Background

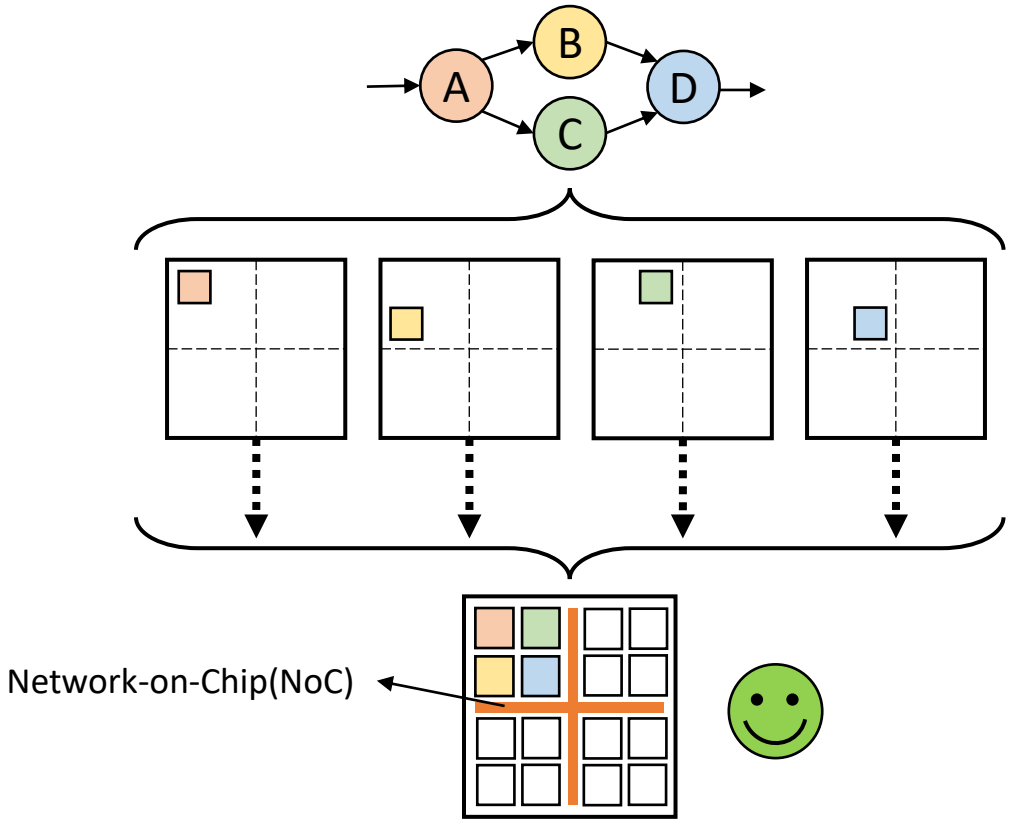
- Recently, researchers suggested
  Separate Compilations in Parallel with Fixed-sized Pages…

  - Case for Fast FPGA Compilation using Partial Reconfiguration, **FPL 2018**
  - Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks, **FPT 2019**
  - Fast linking of separately compiled FPGA blocks without a NoC, **FPT 2020**
  - Software-like compilation for data center fpga accelerators, **HEART 2021**
  - Rapidstream: Parallel physical implementation of FPGA HLS designs , **FPGA 2022**
  - PLD: Fast FPGA compilation to make reconfigurable acceleration compatible with modern incremental refinement software development, **ASPLOS 2022**
  - HiPR: High-level Partial Reconfiguration for Fast Incremental FPGA Compilation, **FPL 2022**

    …

Background

- Recently, researchers suggested
  Separate Compilations in Parallel with Fixed-sized Pages…



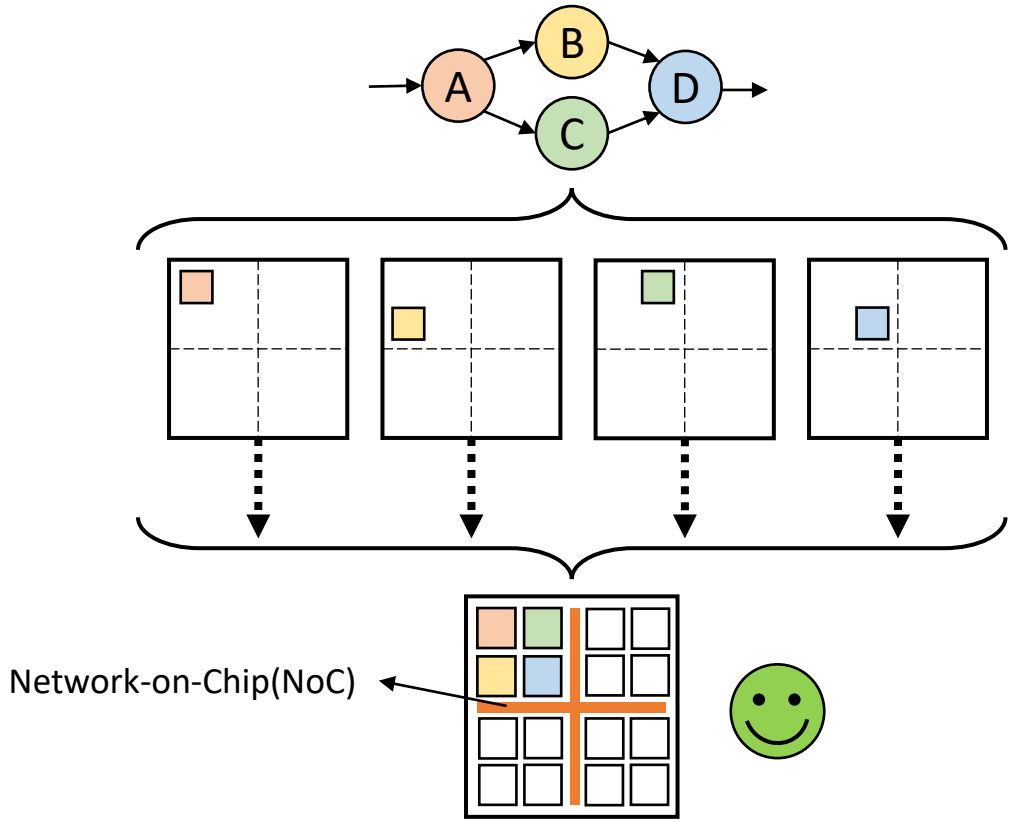
Partially compile FPGA
➔ *Partial Reconfiguration(PR)*

<Advantages>
- **Faster compile (65~110 min → 9~19 min) [1]**
  - **Smaller problem size**
  - **More CPU cores utilized (either local or cloud)**
- **Recompile only modified part**

[1] Xiao et al. "PLD: Fast FPGA compilation to make reconfigurable acceleration compatible with modern incremental refinement software development", ASPLOS 2022

Background

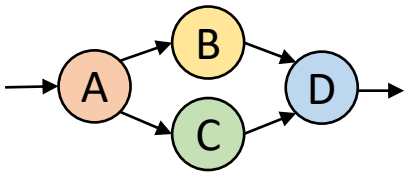
- Recently, researchers suggested
  Separate Compilations in Parallel with Fixed-sized Pages…



So, are we all good? 🤔

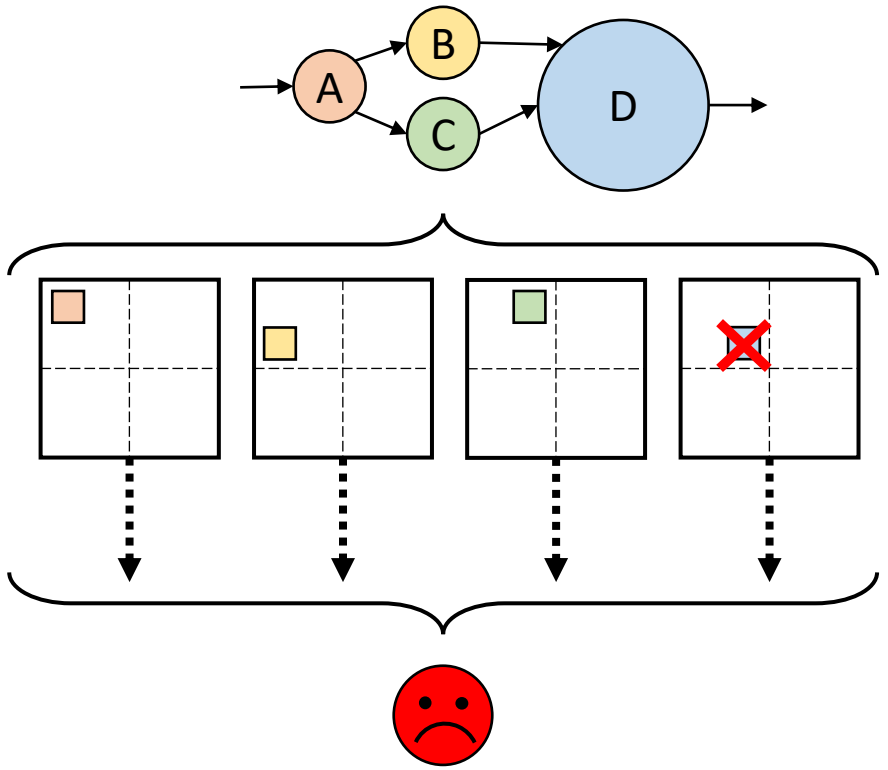
Background

- What if the sizes of operators are unbalanced?



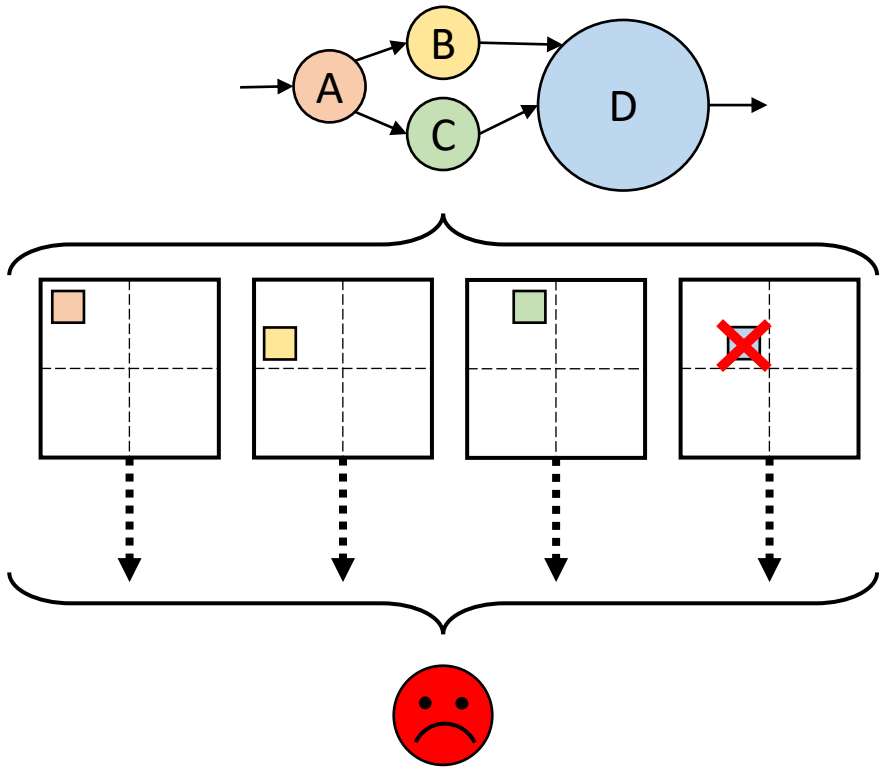
Background

- What if the sizes of operators are unbalanced?



Background

- What if the sizes of operators are unbalanced?
- And what if a user wants to optimize further?



Background

- What if the sizes of operators are unbalanced?
- And what if a user wants to optimize further?

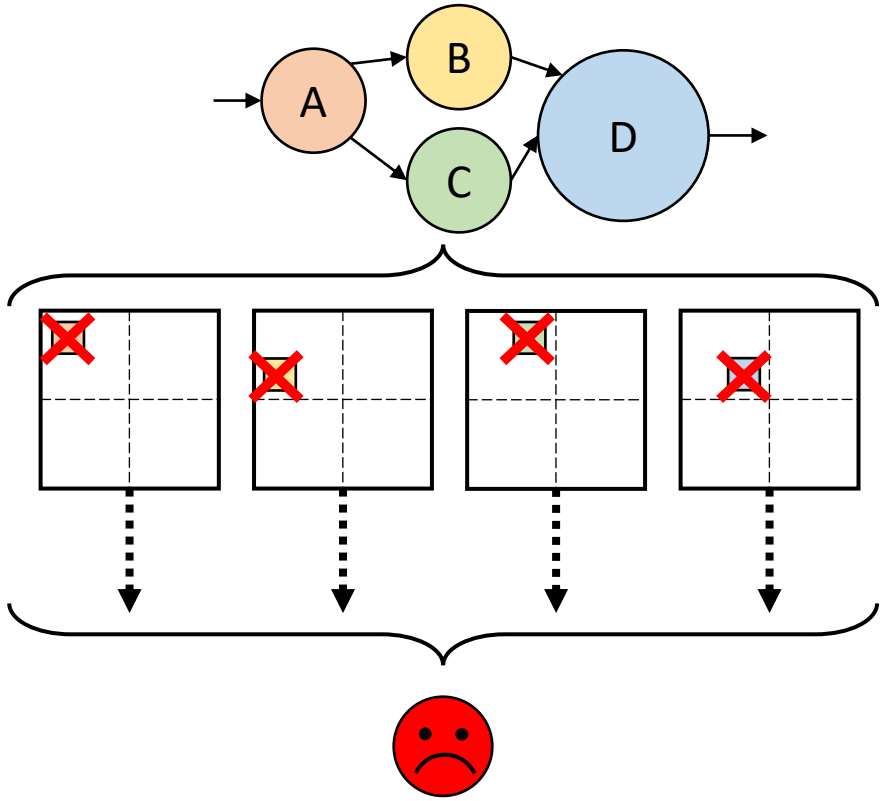
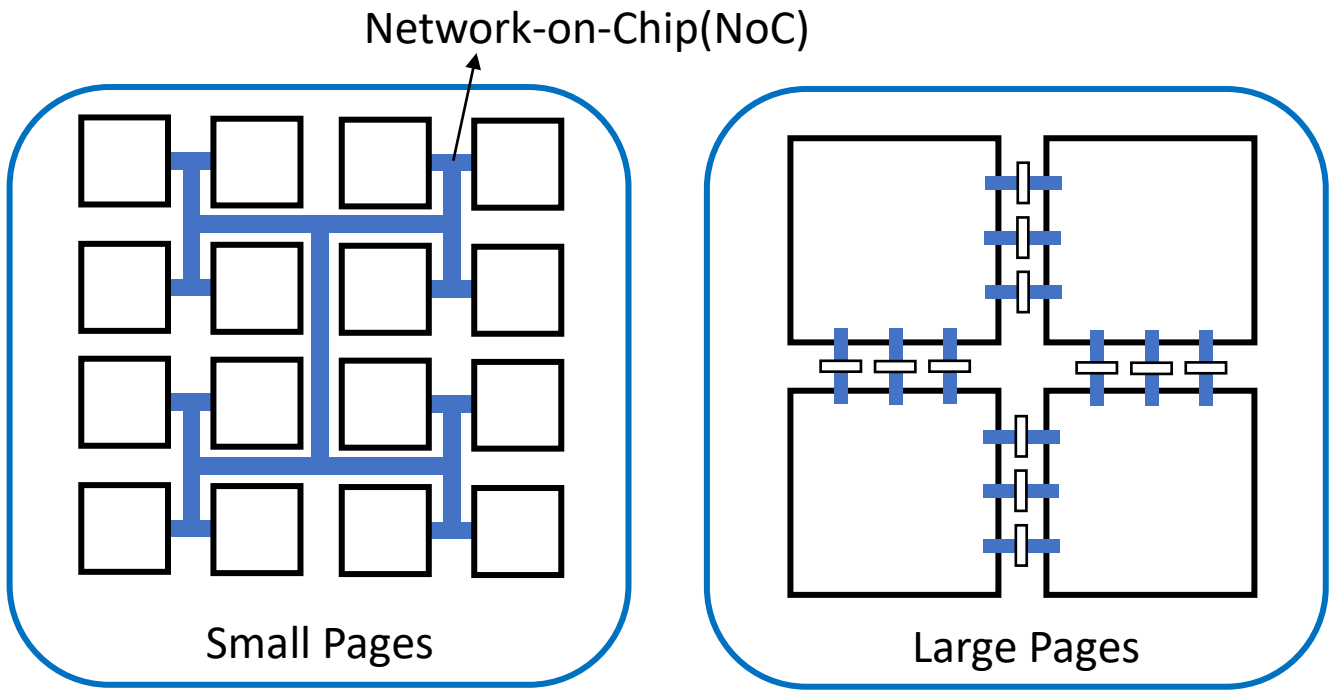


Table of Contents

Table of Contents

Problem

- **Problem is that the pages are fixed-sized**
  1) If the pages are large, it reduces the benefit of separate compilations.
  2) If the pages are small, the users need to manually divide the design into small operators. Also causes NoC bandwidth bottleneck.



Problem

- **Problem is that the pages are fixed-sized**
  1) If the pages are large, it reduces the benefit of separate compilations.
  2) If the pages are small, the users need to manually divide the design into small operators. Also causes NoC bandwidth bottleneck.

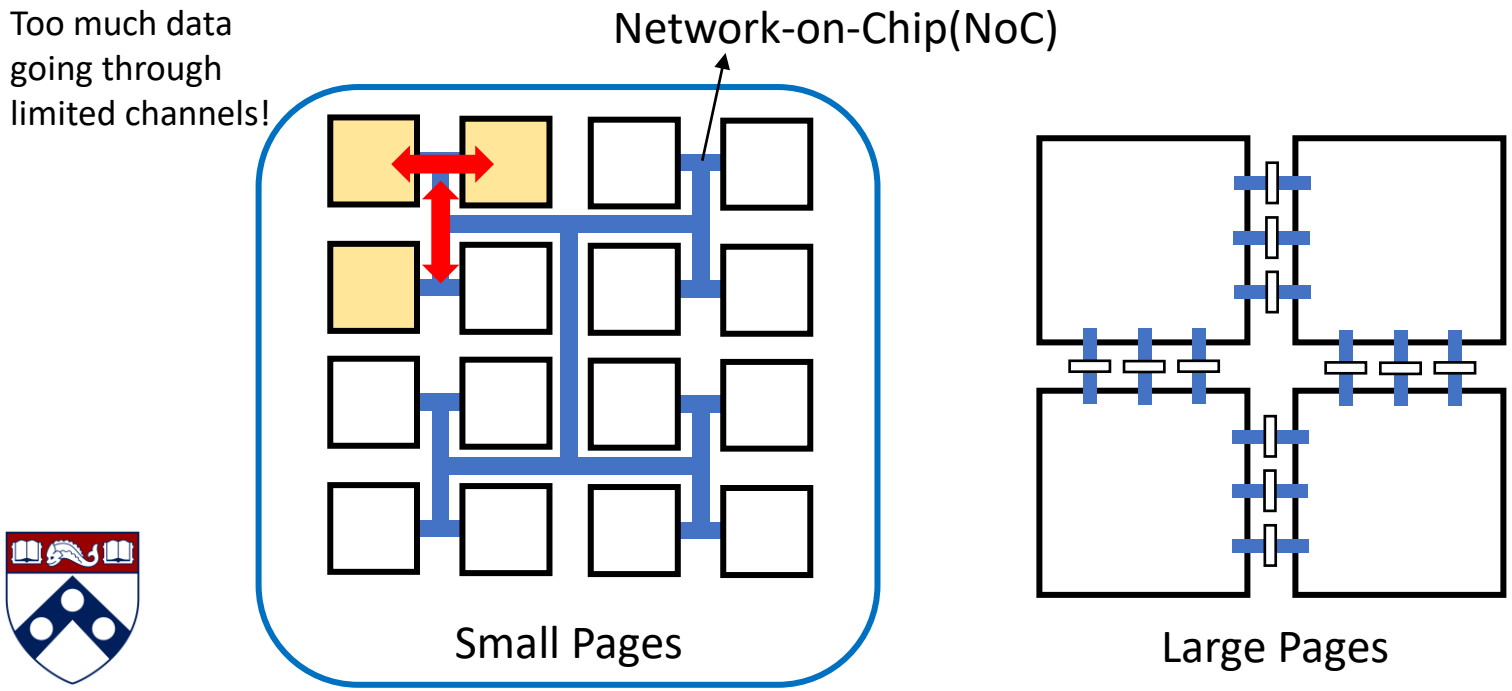
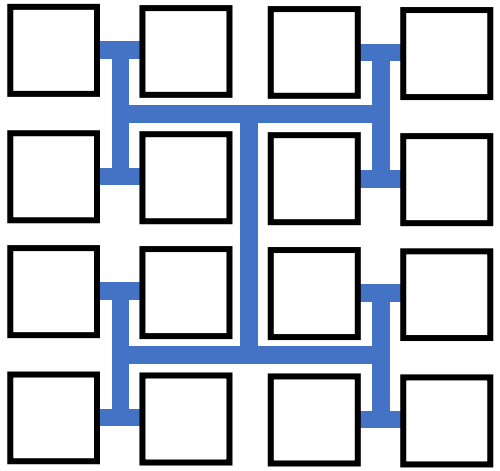


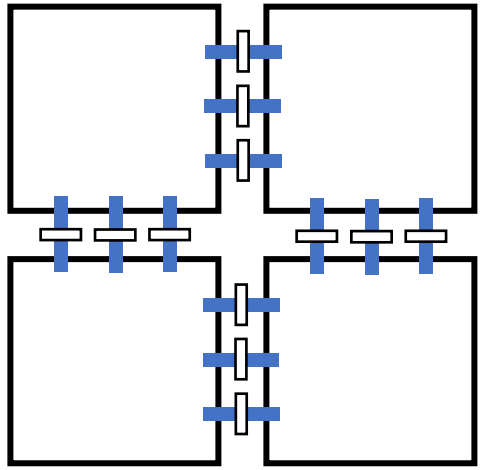
Table of Contents

Idea

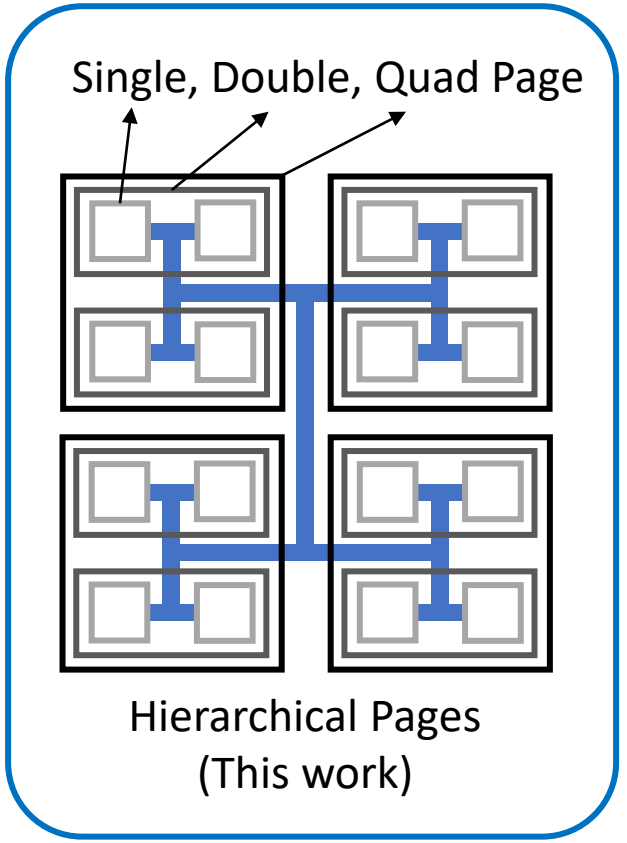
- Separate Compilations in Parallel using *Hierarchical Partial Reconfiguration*
  - A.K.A Nested DFX, recently supported by AMD
  - Partial region inside partial region



Small Pages



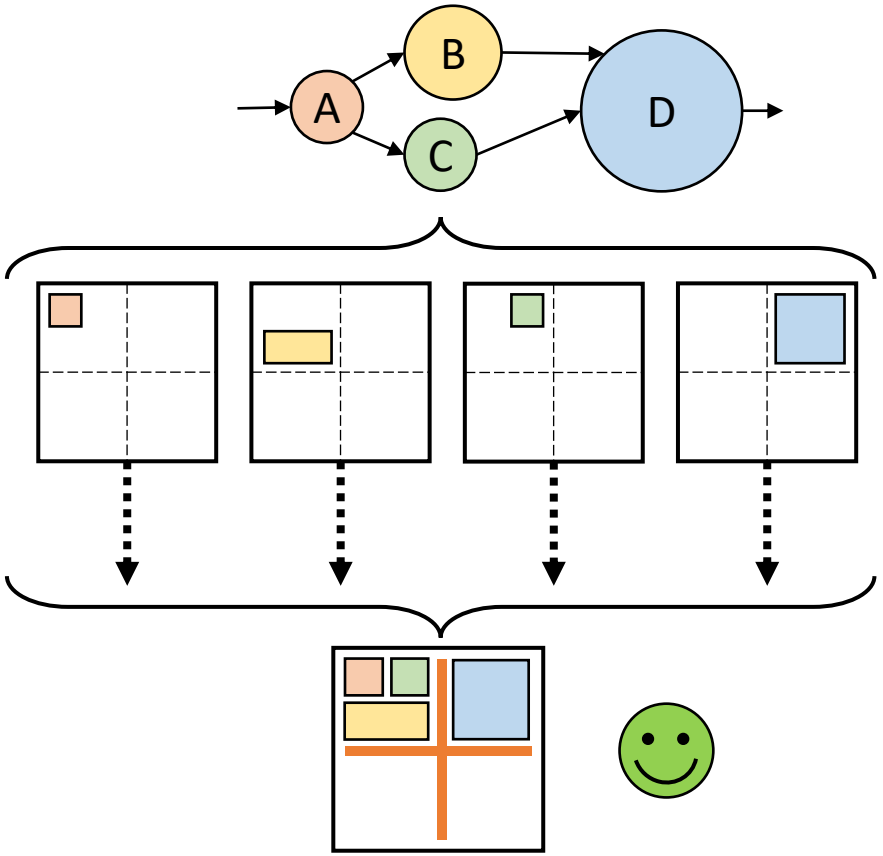
Large Pages



Hierarchical Pages
(This work)

Idea

- Separate Compilations in Parallel using *Hierarchical Partial Reconfiguration*



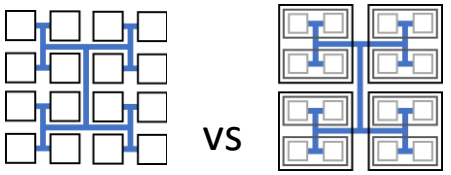
Idea – Fast and Flexible FPGA development using Hierarchical PR

- **1st framework** to exploit Hierarchical Partial Reconfiguration to provide **variable-sized pages**
- Advantages
  - Fine-grained separate compilations with single pages
    ➔ maximize benefits of **fast** separate compilations
  - Users are not forced to decompose a design into small operators. They can use double pages or quad pages.
    ➔ **flexible** framework
  - Useful in incremental development

Table of Contents

Evaluation

- Incremental development scenario with Optical Flow in Rosetta[2]

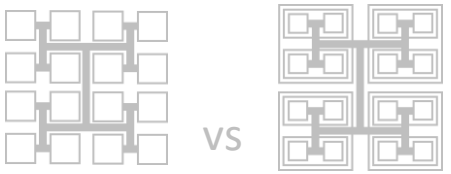


- Performance benefits and compilation improvement
  - Demonstrate our variable-sized pages system achieve **performance improvement** compared to the fixed-sized pages system (1.4~4.9×)
  - Demonstrate our variable-sized pages system **compiles faster** than the vendor tool (2.2~5.3×)

[2] Zhou et al. “Rosetta: A realistic high-level synthesis benchmark suite for software programmable FPGAs”, FPGA 2018

Evaluation

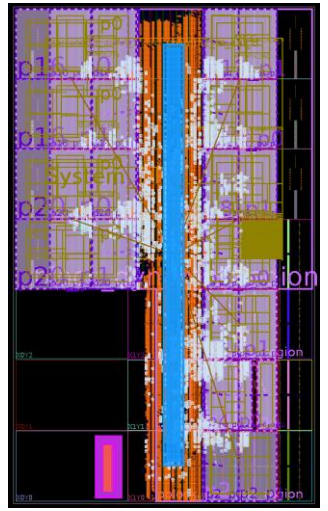
- Incremental development scenario with Optical Flow in Rosetta[2]



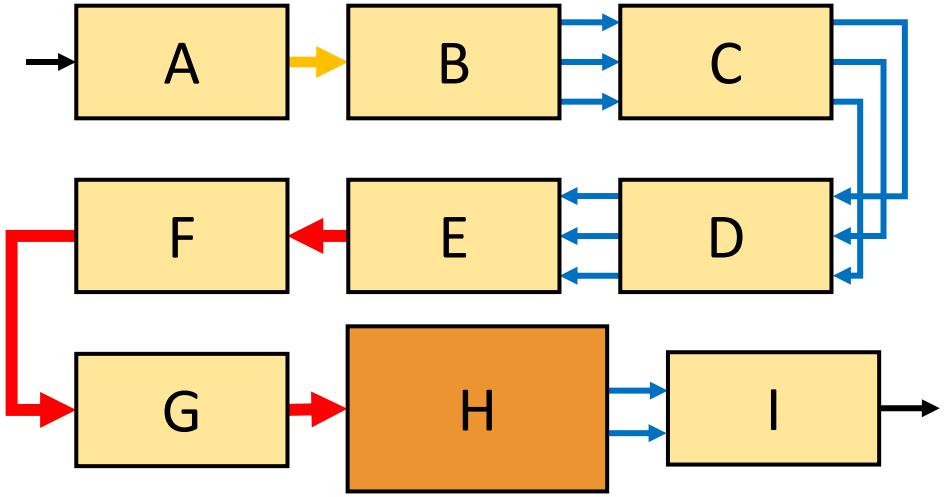
- Performance benefits and compilation improvement
  - Demonstrate our variable-sized pages system achieve **performance improvement** compared to the fixed-sized pages system (1.4~4.9×)
  - Demonstrate our variable-sized pages system **compiles faster** than the vendor tool (2.2~5.3×)

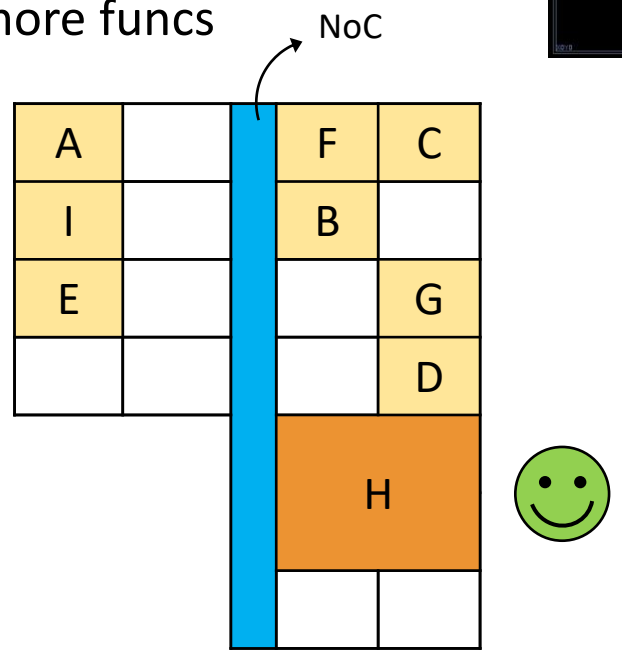
[2] Zhou et al. “Rosetta: A realistic high-level synthesis benchmark suite for software programmable FPGAs”, FPGA 2018

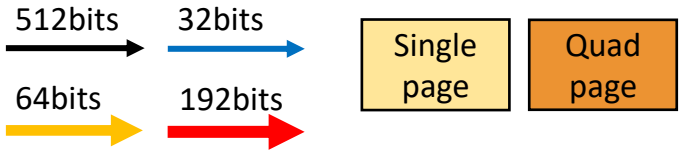
Evaluation



- Incremental development scenario
  - **Quickly start** from the *natural decomposition*!
    - You want to quickly try on FPGA and incrementally add more funcs



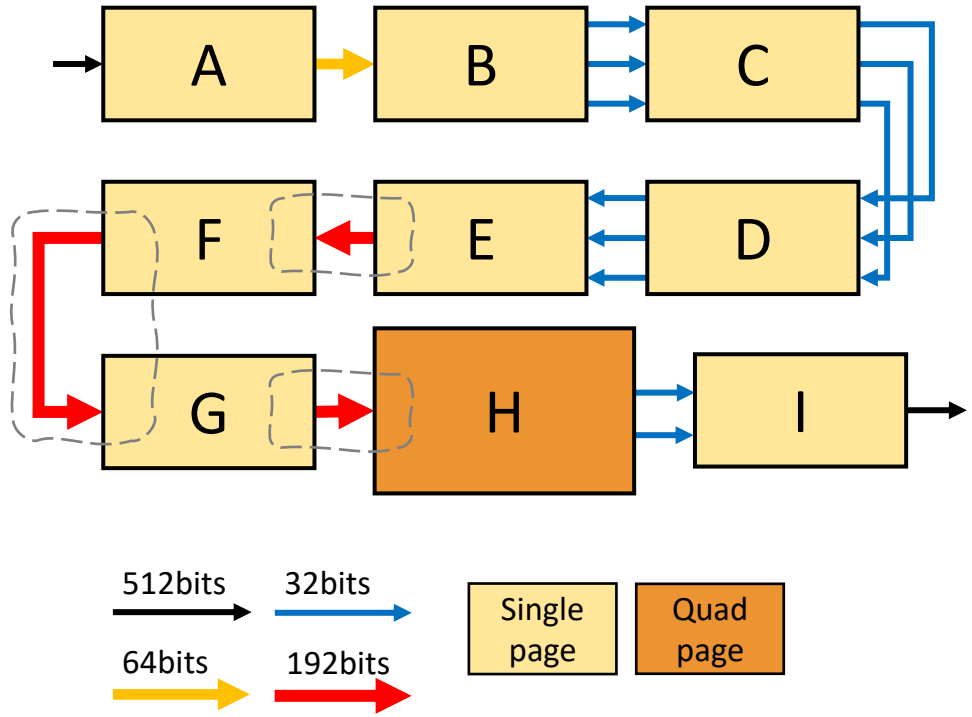


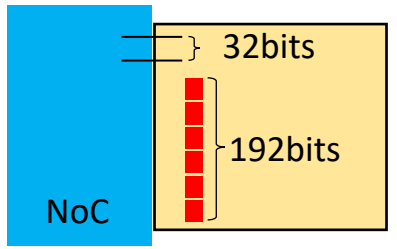


<FPGA mapping>

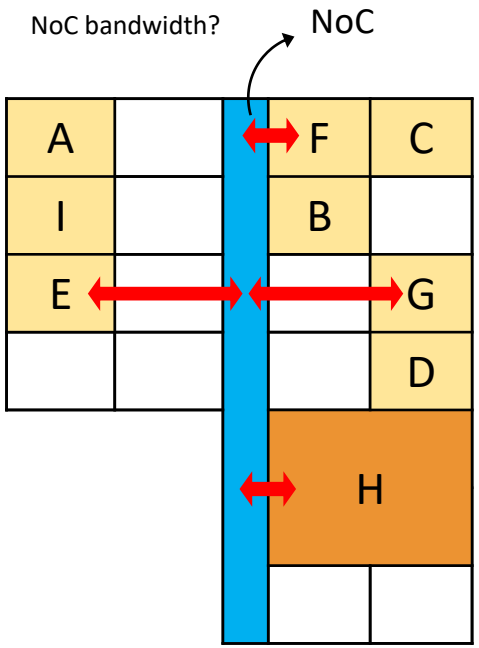
Evaluation

- Incremental development scenario

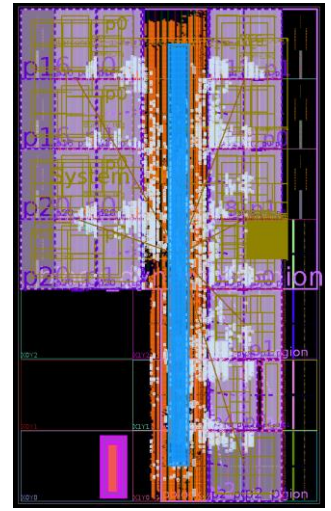




NoC bandwidth?

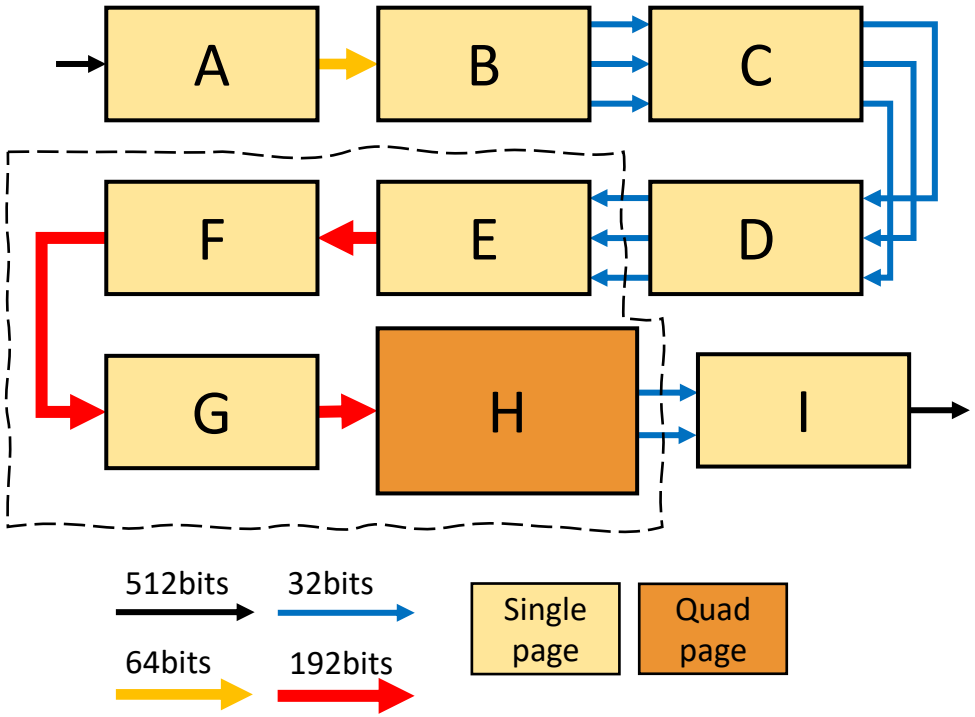


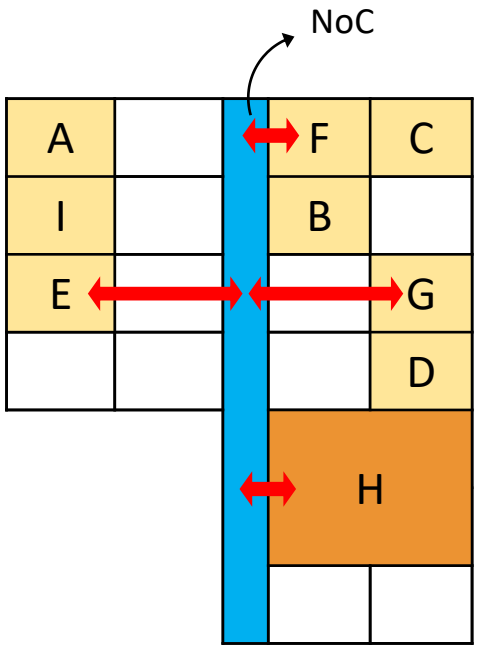
<FPGA mapping>



Evaluation

- Incremental development scenario
  - **Merge** operators to remove NoC bandwidth bottleneck

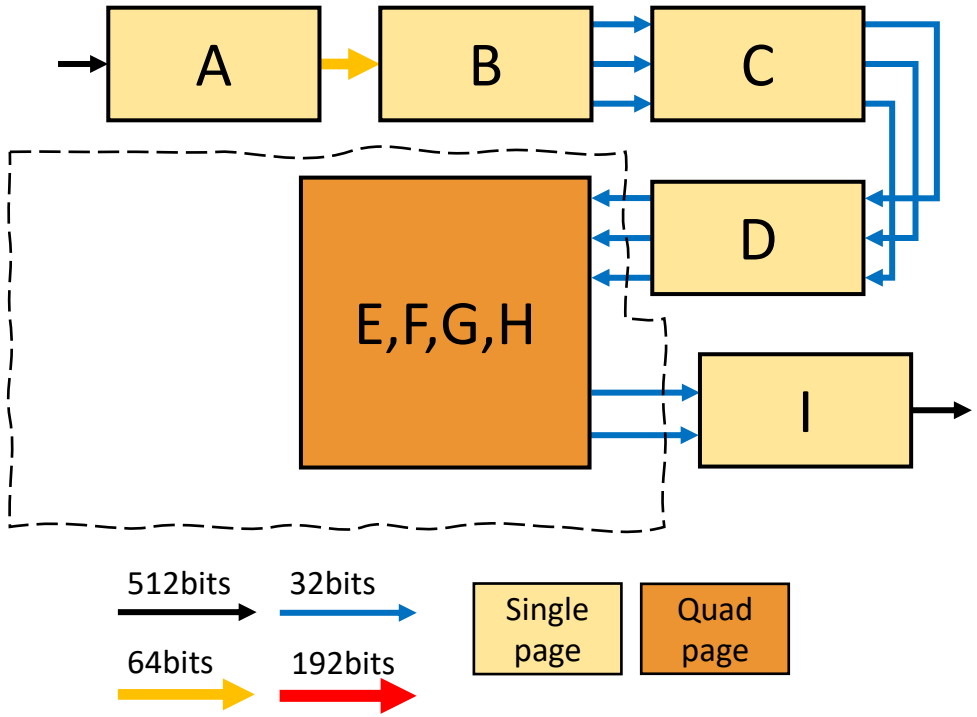


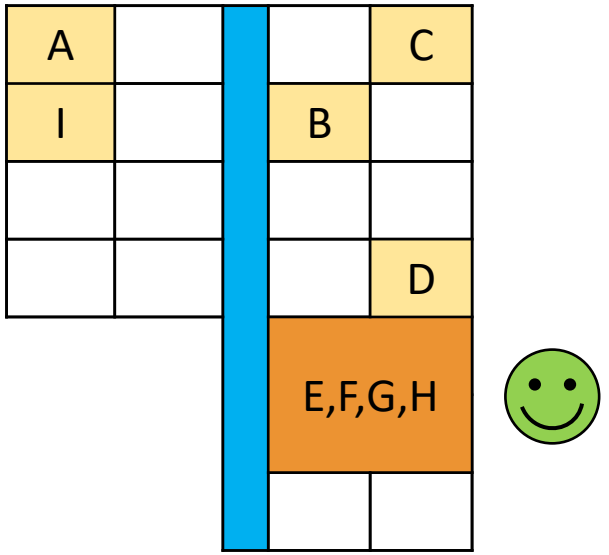


<FPGA mapping>

Evaluation

- Incremental development scenario
  - **Merge** operators to remove NoC bandwidth bottleneck

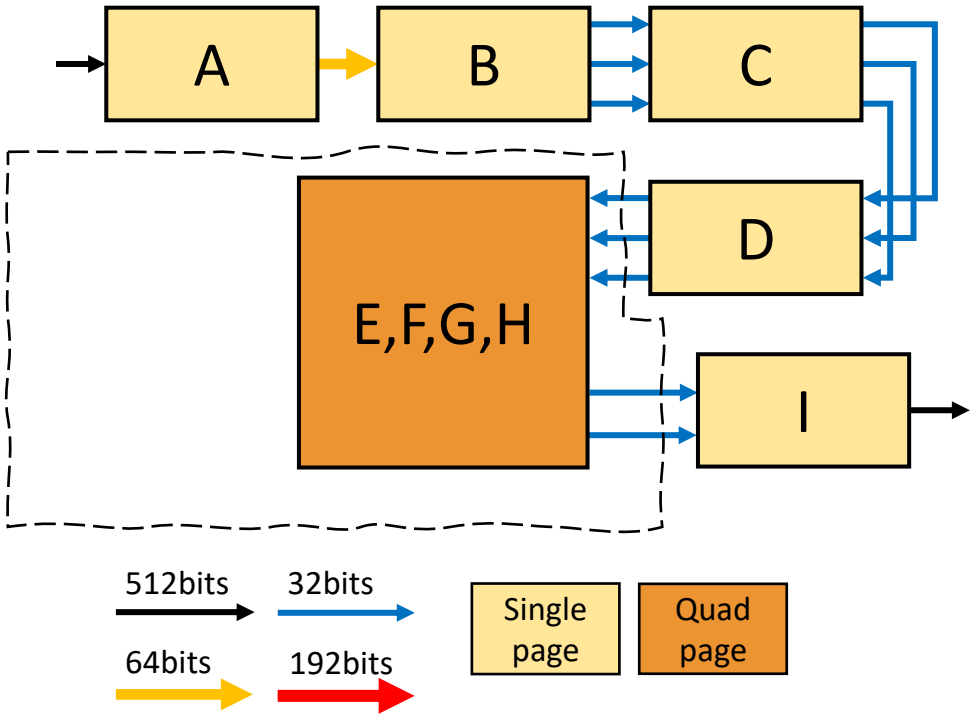




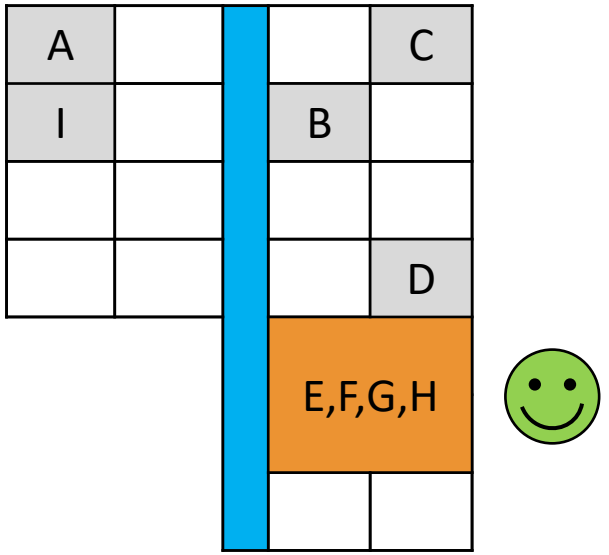
<FPGA mapping>

Evaluation

- Incremental development scenario
  - **Merge** operators to remove NoC bandwidth bottleneck



Note that only necessary pages are recompiled



<FPGA mapping>

Evaluation

- Incremental development scenario with Optical Flow
  - Followed by other optimizations (please look at our paper for details)

<Vendor tool>

	Monolithic Compile Time
`calc_pixel_t: 64b` Ver. 1	711s
`calc_pixel_t: 96b` Ver. 2	695s

<Incremental Development with our framework>

steps	Largest Page	Incremental Compile Time
1	Quad	261s

Evaluation

- Incremental development scenario with Optical Flow
  - Followed by other optimizations (please look at our paper for details)

<Vendor tool>

	Monolithic Compile Time
calc_pixel_t: 64b	Ver. 1	711s
calc_pixel_t: 96b	Ver. 2	695s

<Incremental Development with our framework>

steps	Largest Page	Incremental Compile Time
1	Quad	261s
2	Quad	245s

Evaluation

- Incremental development scenario with Optical Flow
  - Followed by other optimizations (please look at our paper for details)

<Vendor tool>

	Monolithic Compile Time
calc_pixel_t: 64b	Ver. 1	711s
calc_pixel_t: 96b	Ver. 2	695s

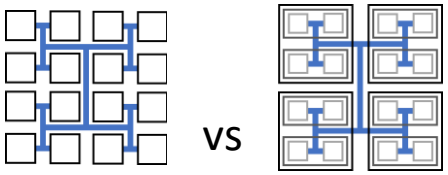
<Incremental Development with our framework>

steps	Largest Page	Incremental Compile Time
1	Quad	261s
2	Quad	245s
3	Quad	274s
4	Quad	288s
5	Single	94s
6	Single	107s
7	Quad	291s
8	Quad	274s

+ Can quickly edit/compile/optimize with our framework
+ For single page, it takes 1~2 min!

Evaluation

- Incremental development scenario with Optical Flow in Rosetta[2]

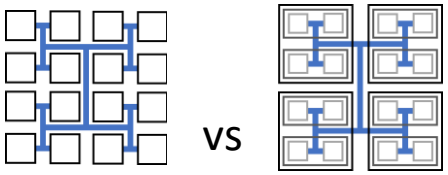


- Performance benefits and compilation improvement
  - Demonstrate our variable-sized pages system achieve **performance improvement** compared to the fixed-sized pages system (1.4~4.9×)
  - Demonstrate our variable-sized pages system **compiles faster** than the vendor tool (2.2~5.3×)

[2] Zhou et al. “Rosetta: A realistic high-level synthesis benchmark suite for software programmable FPGAs”, FPGA 2018

Evaluation

- Incremental development scenario with Optical Flow in Rosetta[2]



- Performance benefits and compilation improvement
  - Demonstrate our variable-sized pages system achieve **performance improvement** compared to the fixed-sized pages system (1.4~4.9×)
  - Demonstrate our variable-sized pages system **compiles faster** than the vendor tool (2.2~5.3×)

[2] Zhou et al. “Rosetta: A realistic high-level synthesis benchmark suite for software programmable FPGAs”, FPGA 2018

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	17	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	17	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	1	322s	2.2×	8.8ms	3.6×
Optical, ver.2	9	0	1	305s	2.3×	8.8ms	4.9×
Rendering, Par=1	5	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	1	0	169s	2.5×	1.8ms	1.4×
Digit Recognition, Par=40	10	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	10	0	251s	5.3×	4.7ms	1.5×

Par = Parallelization Factor

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	17	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	17	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	1	322s	2.2×	8.8ms	3.6×
Optical, ver.2	9	0	1	305s	2.3×	8.8ms	4.9×
Rendering, Par=1	5	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	1	0	169s	2.5×	1.8ms	1.4×
Digit Recognition, Par=40	10	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	10	0	251s	5.3×	4.7ms	1.5×

Par = Parallelization Factor

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	**17**	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	**17**	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	1	322s	2.2×	8.8ms	3.6×
Optical, ver.2	9	0	1	305s	2.3×	8.8ms	4.9×
Rendering, Par=1	**5**	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	1	0	169s	2.5×	1.8ms	1.4×
Digit Recognition, Par=40	**10**	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	10	0	251s	5.3×	4.7ms	1.5×

Par = Parallelization Factor

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	17	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	17	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	1	322s	2.2×	8.8ms	3.6×
Optical, ver.2	9	0	1	305s	2.3×	8.8ms	4.9×
Rendering, Par=1	5	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	1	0	169s	2.5×	1.8ms	1.4×
Digit Recognition, Par=40	10	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	10	0	251s	5.3×	4.7ms	1.5×

Par = Parallelization Factor

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	17	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	17	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	**1**	322s	2.2×	8.8ms	3.6×
Optical, ver.2	9	0	**1**	305s	2.3×	8.8ms	4.9×
Rendering, Par=1	5	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	**1**	0	169s	2.5×	1.8ms	1.4×
Digit Recognition, Par=40	10	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	**10**	0	251s	5.3×	4.7ms	1.5×

Par = Parallelization Factor

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	17	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	17	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	1	322s	2.2×	8.8ms	3.6×
Optical, ver.2	9	0	1	305s	2.3×	8.8ms	4.9×
Rendering, Par=1	5	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	1	0	169s	2.5×	1.8ms	1.4×
Digit Recognition, Par=40	10	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	10	0	251s	5.3×	4.7ms	1.5×

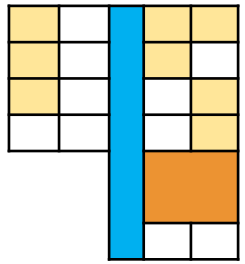
Par = Parallelization Factor

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	17	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	17	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	1	322s	**2.2×**	8.8ms	**3.6×**
Optical, ver.2	9	0	1	305s	**2.3×**	8.8ms	**4.9×**
Rendering, Par=1	5	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	1	0	169s	2.5×	1.8ms	1.4×
Digit Recognition, Par=40	10	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	10	0	251s	5.3×	4.7ms	1.5×

Par = Parallelization Factor



Improvement in performance thanks to

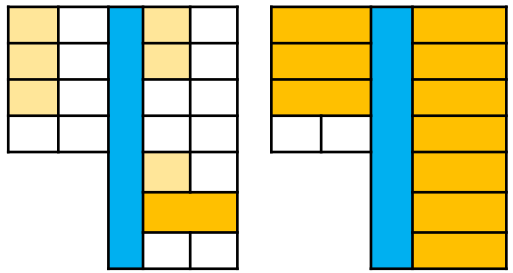
1) **Removing NoC bandwidth bottleneck by merging ops**
2) Use more area for single operator(double, quad page)

Evaluation

<Benchmarks with PR pages>

Benchmarks	Usage by Page Size			Compile Time	Compile Speedup over the vendor tool	App Latency	App Latency over Fixed-Sized Page
	Single	Double	Quad				
Optical, ver.1, fixed-sized	17	0	0	276s	n/a	32.1ms	1.0×
Optical, ver.2, fixed-sized	17	0	0	329s	2.2x	43.4ms	1.0×
Optical, ver.1	9	0	1	322s	2.2×	8.8ms	3.6×
Optical, ver.2	9	0	1	305s	2.3×	8.8ms	4.9×
Rendering, Par=1	5	0	0	151s	2.8×	2.6ms	1.0×
Rendering, Par=2	6	1	0	169s	**2.5×**	1.8ms	**1.4×**
Digit Recognition, Par=40	10	0	0	212s	4.3×	7.0ms	1.0×
Digit Recognition, Par=80	0	10	0	251s	**5.3×**	4.7ms	**1.5×**

Par = Parallelization Factor



Improvement in performance thanks to

1) Removing NoC bandwidth bottleneck by merging ops
2) **Use more area for single operator(double, quad page)**

Table of Contents

Conclusion

Available,
Evaluated Functional,
Evaluated Reusable,
Results Replicated

- An open-source framework (https://github.com/icgrp/prflow_nested_dfx)
- **Variable-sized pages** using hierarchical partial reconfiguration
  - Gives 1.4~4.9× performance improvement compared to the fixed-sized pages
  - While compiling 2.2~5.5× faster than the commercial tool(AMD-Xilinx)
- In the incremental development scenario, a single page takes **1~2 min** and a quad page takes **4~5 min** for the design that the vendor takes **11~12 min**

Thank you ☺