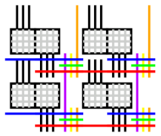
Fast and Flexible FPGA development

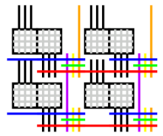
Dongjoon(DJ) Park
dopark@seas.upenn.edu



Implementation of Computation Group
University of Pennsylvania

Fast and Flexible FPGA ? development

Dongjoon(DJ) Park
dopark@seas.upenn.edu



Implementation of Computation Group
University of Pennsylvania

Background

How many of you have heard of “**FPGAs**”?

Background

How many of you have heard of "**FPGAs**"?

Maybe in Digital Systems course?

In Computer Architecture course?

Background

- FPGA?
  - Field-Programmable Gate Arrays

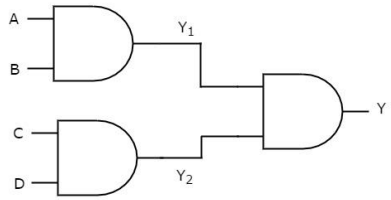
High-level C code

```
int a = b + c
…
```

Hardware Description Language (HDL)

```
assign a = b + c
…
```

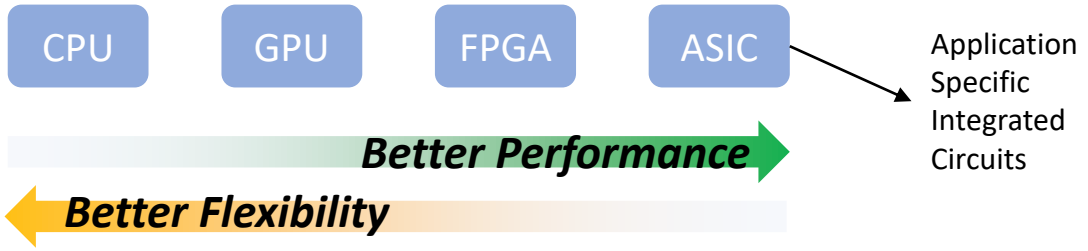
Gate level representation



Mapped on FPGA device

Background

- Comparison – Let's take it as granted for now...



- Deployed in a wide range of applications
  - Machine learning, image processing, data analysis, etc

Background

Once, a friend of mine asked,

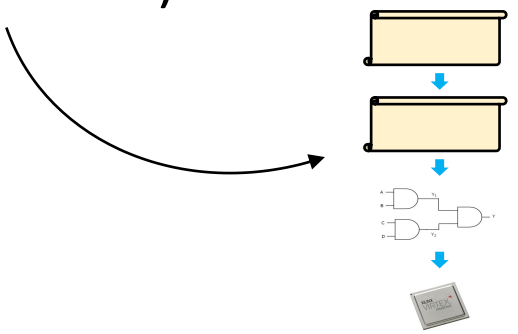
“Then, why not using FPGAs everywhere?”

Table of Contents

Table of Contents

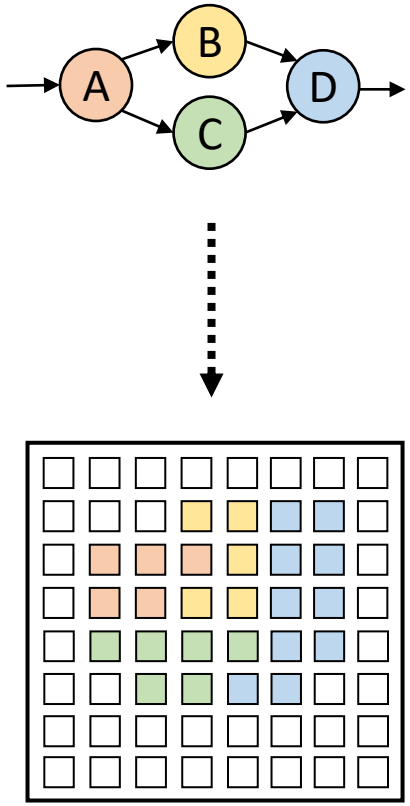
Problem

- FPGA *compilation* is slow
  - Compilation: the process of a user code is translated in a form that the HW can interpret (the process I showed in the first slide)
  - C code example: `gcc test.c`
- 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…



Problem

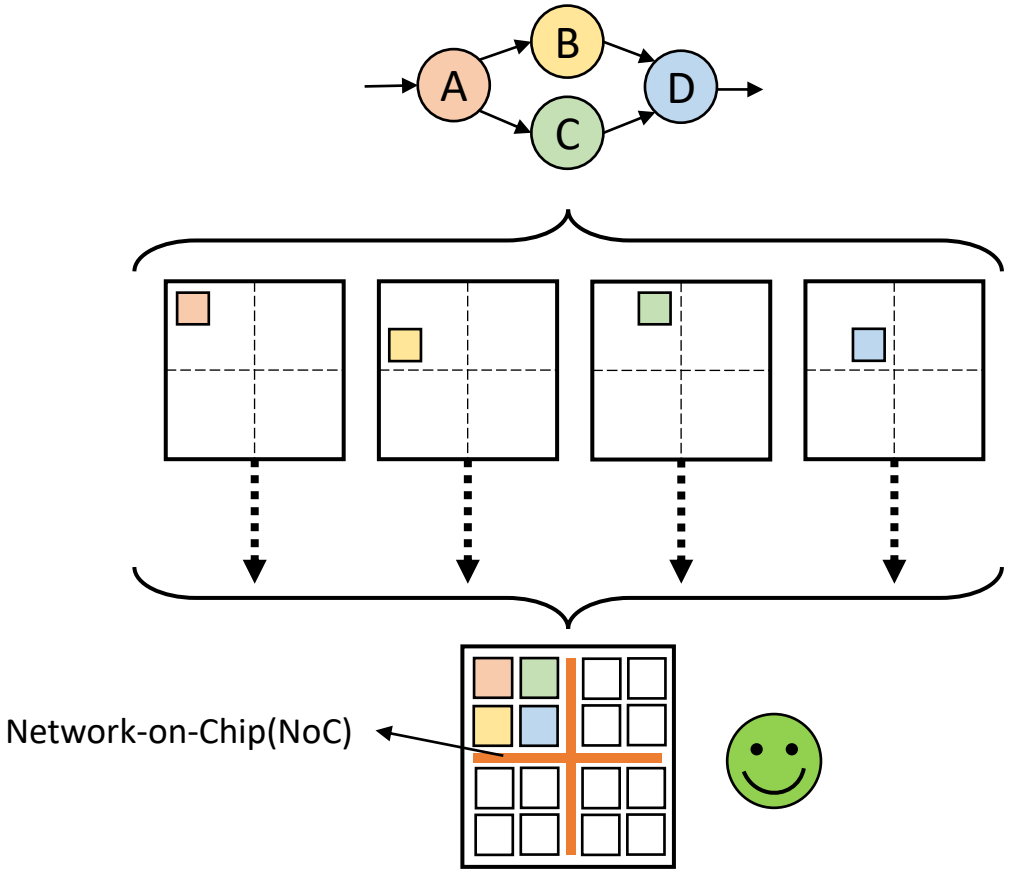
- Why slow?: FPGA's compilation on vendor tool(Intel, AMD-Xilinx) is *monolithic*



Resolution? 🤔

Problem

- Recently, researchers suggest *Separate Compilations in Parallel*



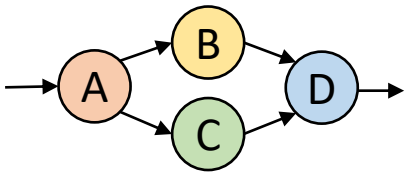
- Case for Fast FPGA Compilation using Partial Reconfiguration, FPL 2018
- Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks, FPT 2019
- HiPR: High-level Partial Reconfiguration for Fast Incremental FPGA Compilation, FPL 2022

Partially compile FPGA
➔ *Partial Reconfiguration(PR)*

<Advantages>
- **Faster compile**
  - **Smaller problem size**

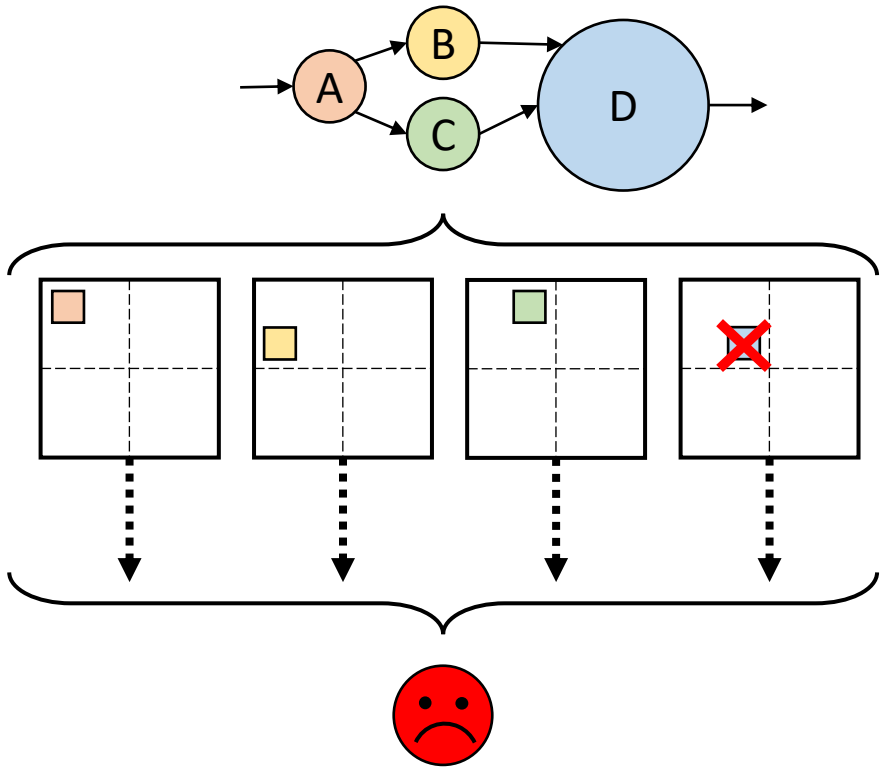
Background

- What if the sizes of operators are unbalanced?



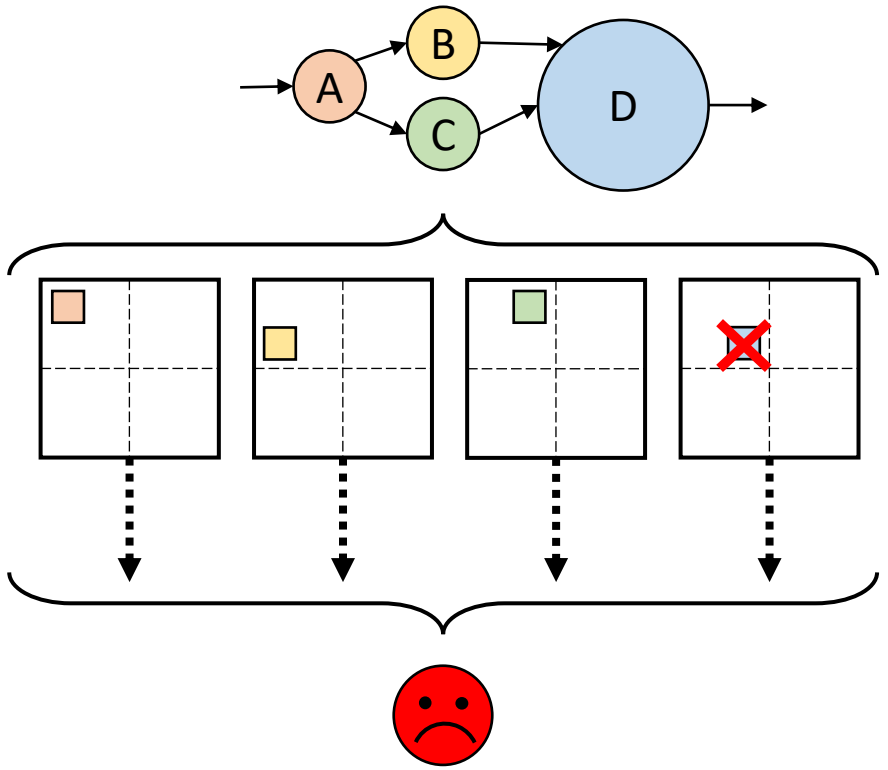
Background

- What if the sizes of operators are unbalanced?



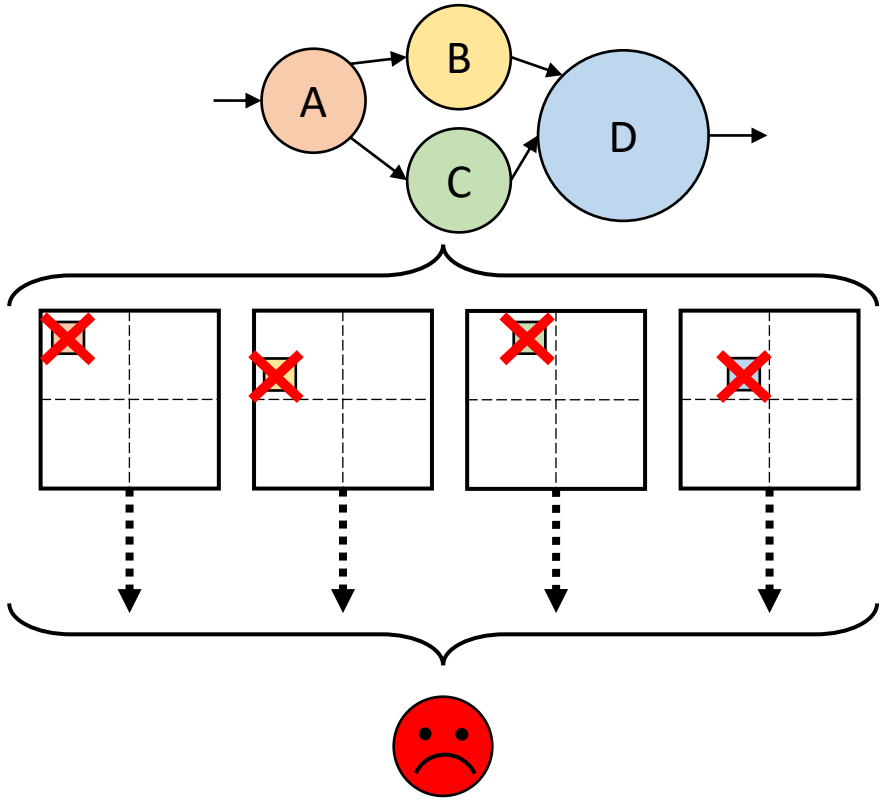
Background

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



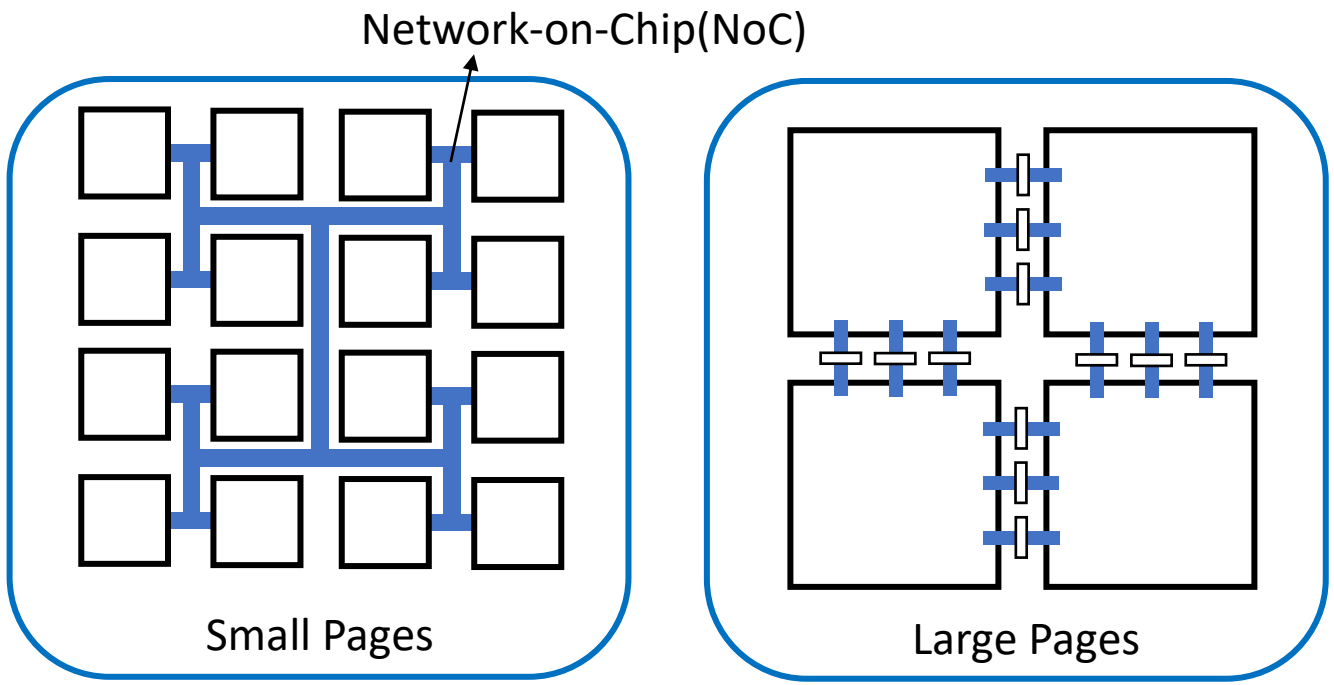
Background

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



Problem that this work tries to solve

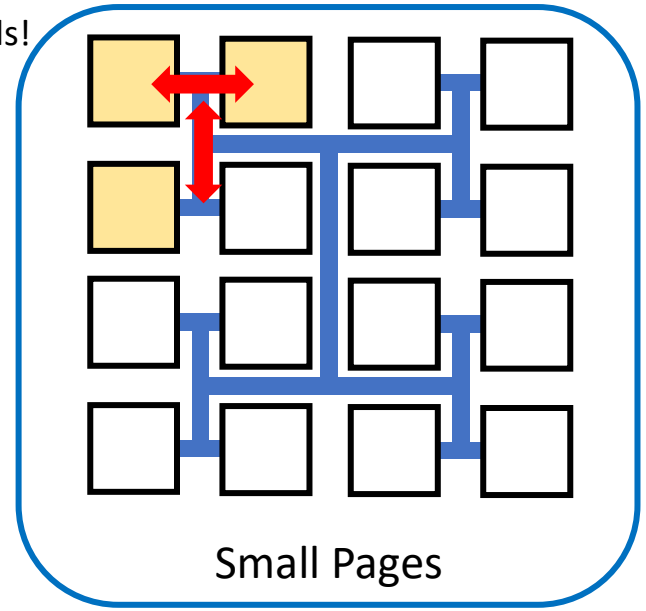
- **Problem is that the *pages* are fixed-sized**
  1) If the pages are large, it reduces the benefit of separate compilations.
  2) If the pages are small, the users need to manually divide the design into small operators. Also causes NoC bandwidth bottleneck



Problem that this work tries to solve

- **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

Too much data going through limited channels!



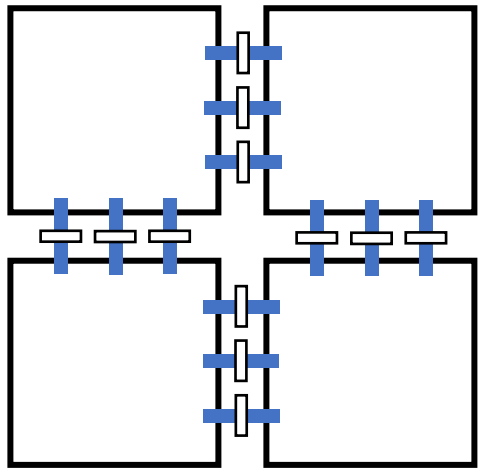
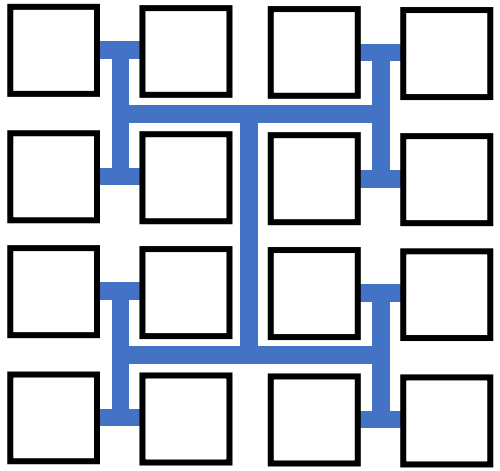


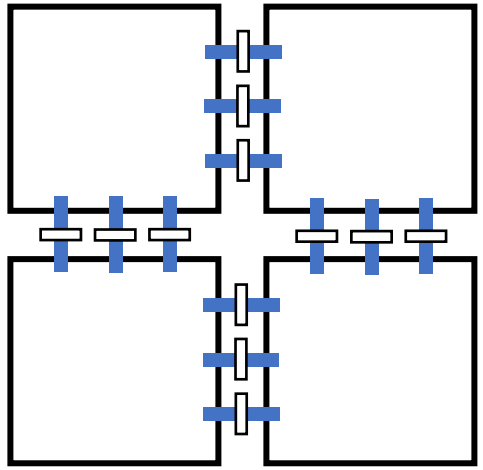
Table of Contents

Idea

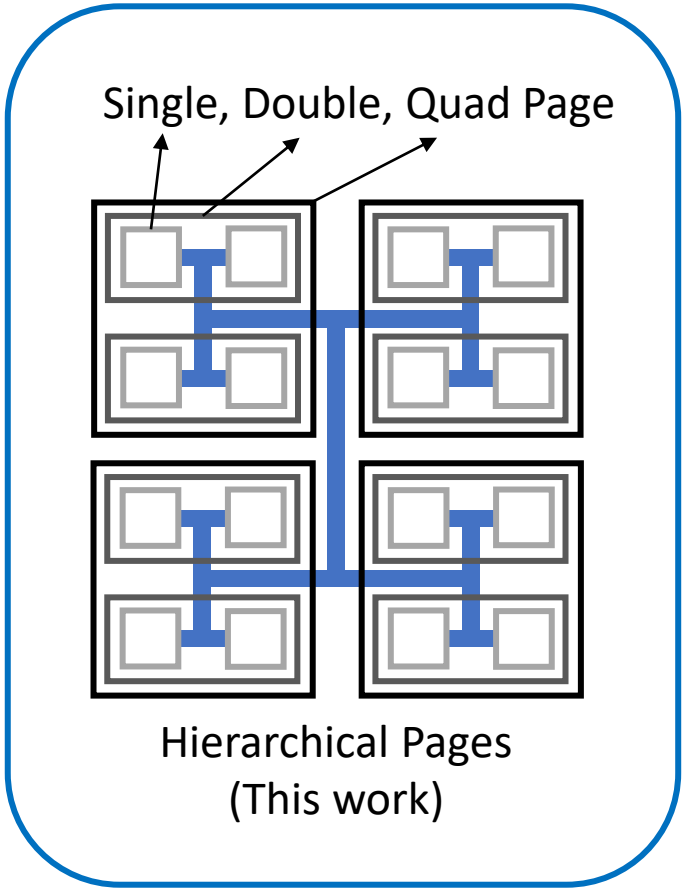
- Separate Compilations in Parallel using *Hierarchical Partial Reconfiguration*
  - Recently supported by AMD-Xilinx
  - Partial region inside a partial region



Small Pages



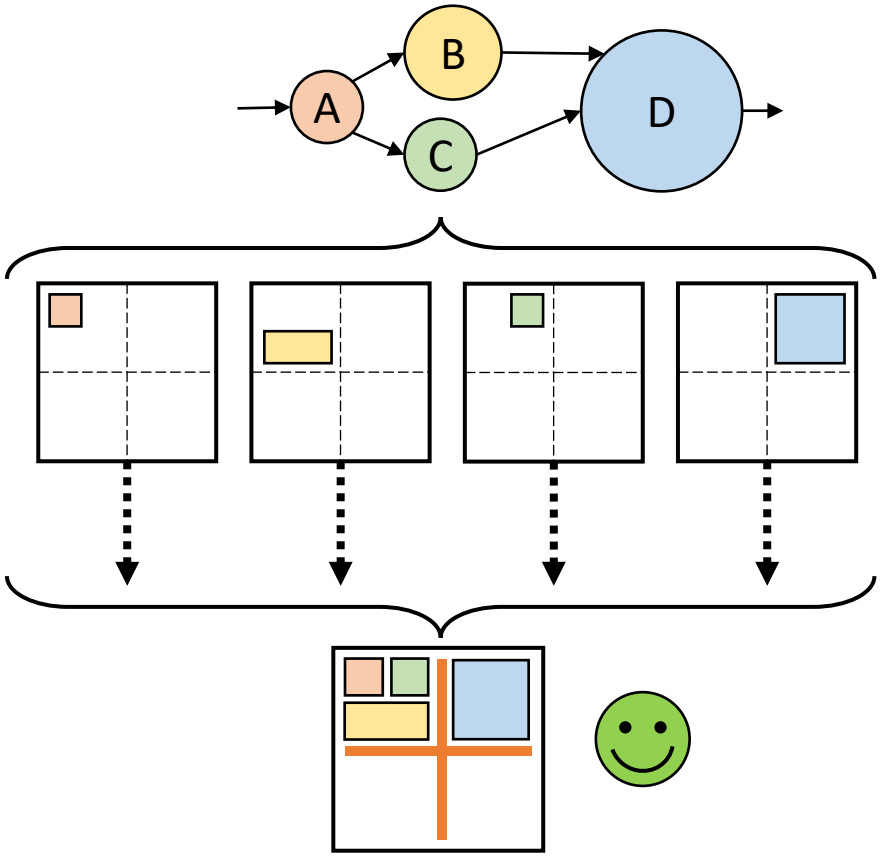
Large Pages



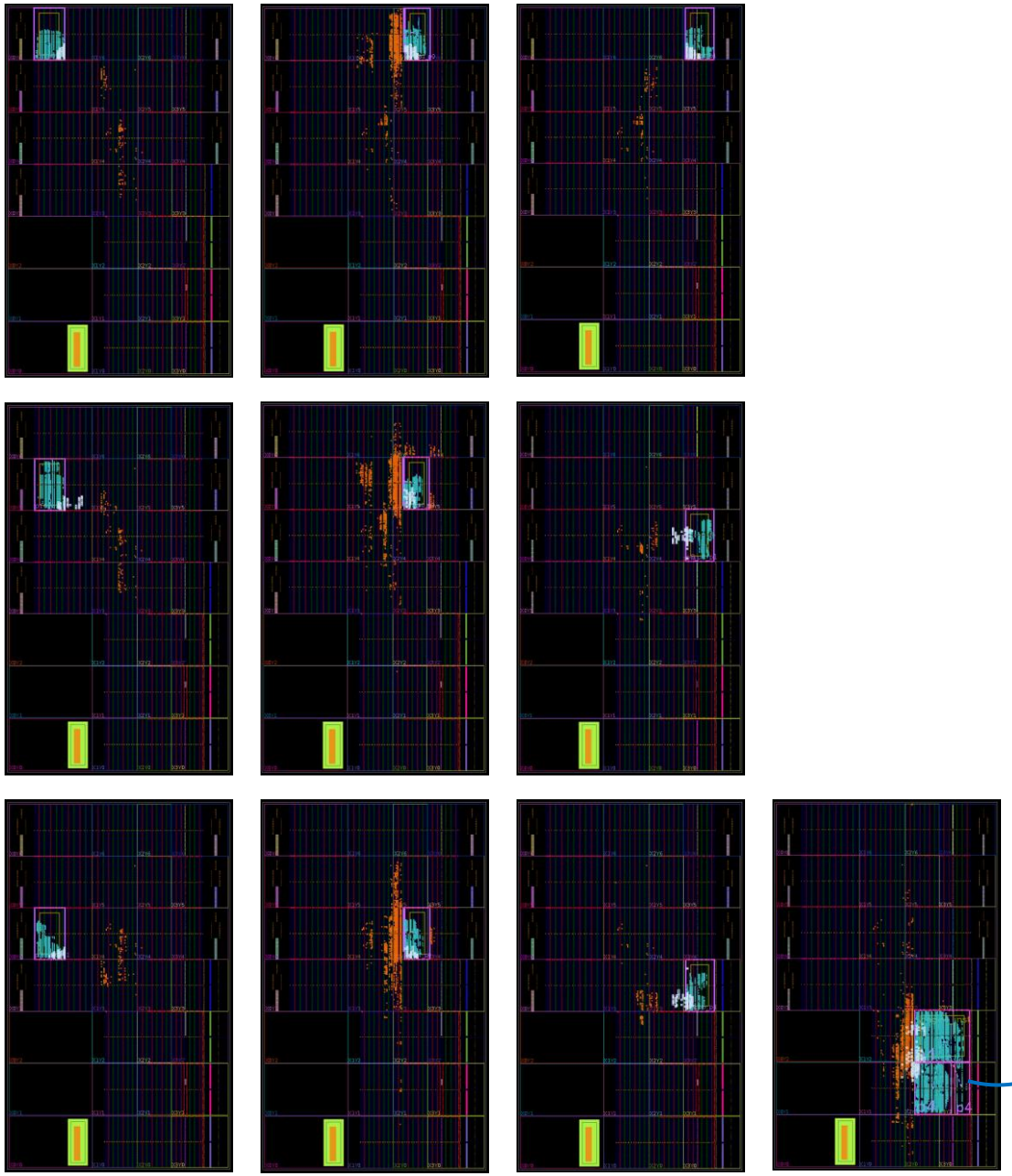
Hierarchical Pages
(This work)

Idea

- Separate Compilations in Parallel using *Hierarchical Partial Reconfiguration*



Idea



Idea

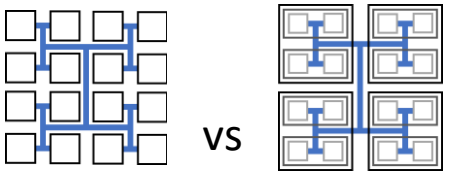
- 1st framework to exploit Hierarchical Partial Reconfiguration to provide **variable-sized pages**
- Advantages
  - Fine-grained separate compilations with single pages
    ➔ maximize benefits of **fast** separate compilations
  - Users are not forced to decompose a design into small operators. They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

Table of Contents

Evaluation

- Incremental development scenario

- Performance benefits and compilation improvement

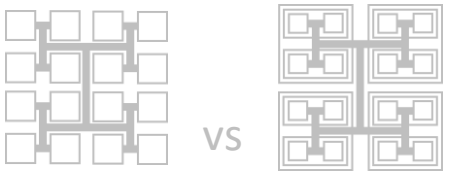


  - Demonstrate our variable-sized pages system achieve **performance improvement** compared to the fixed-sized pages system (1.4~4.9×)
  - Demonstrate our variable-sized pages system **compiles faster** than the vendor tool (2.3~5.5×)

Evaluation

- Incremental development scenario

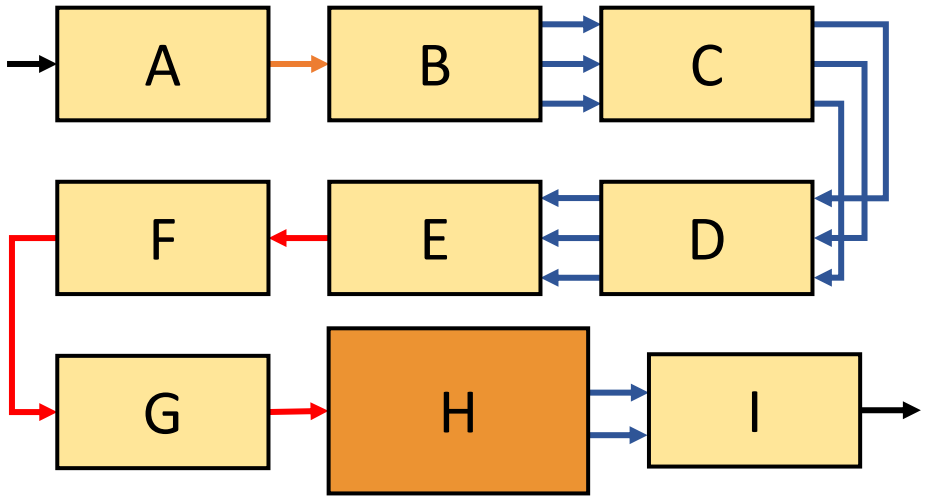
- Performance benefits and compilation improvement

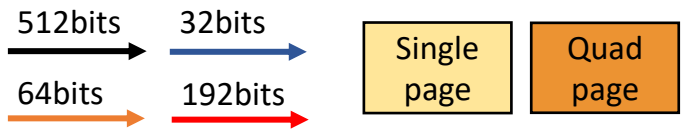


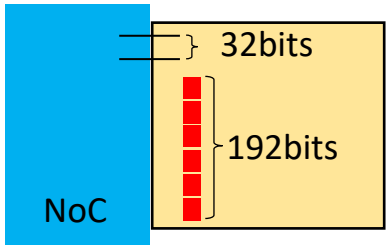
  - Demonstrate our variable-sized pages system achieve **performance improvement** compared to the fixed-sized pages system (1.4~4.9×)
  - Demonstrate our variable-sized pages system **compiles faster** than the vendor tool (2.3~5.5×)

Evaluation

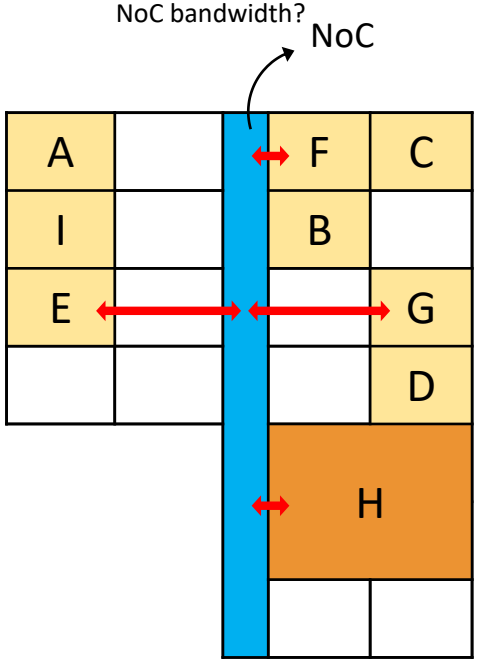
- Incremental development scenario
  - **Quickly start** from natural decomposition!



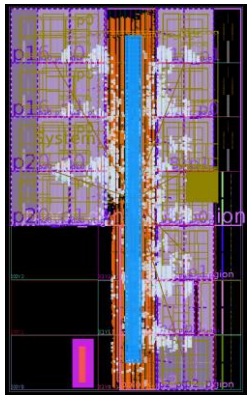




NoC bandwidth?

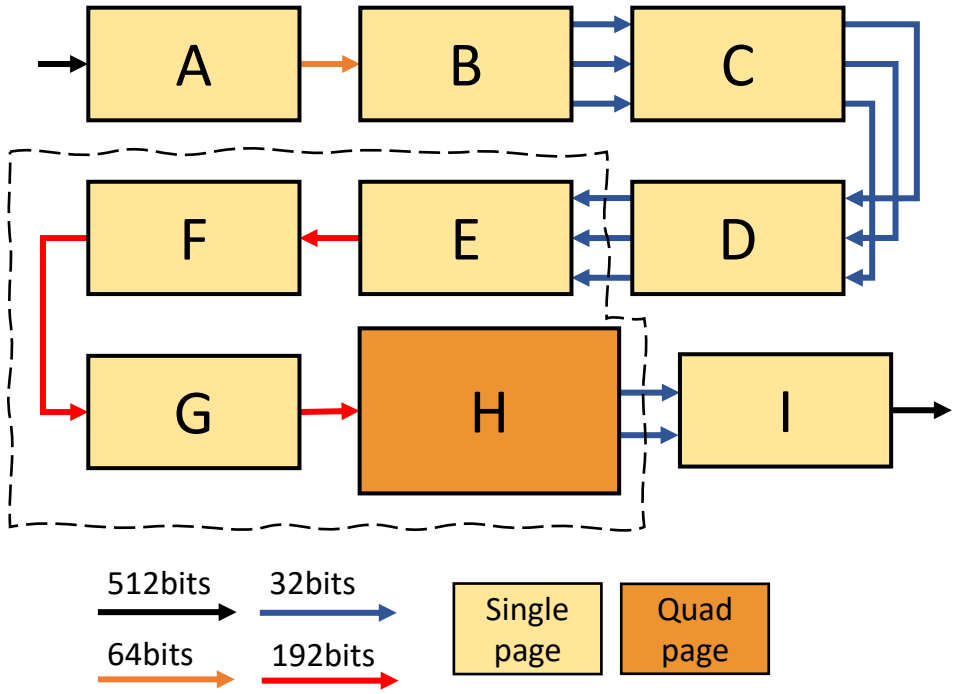


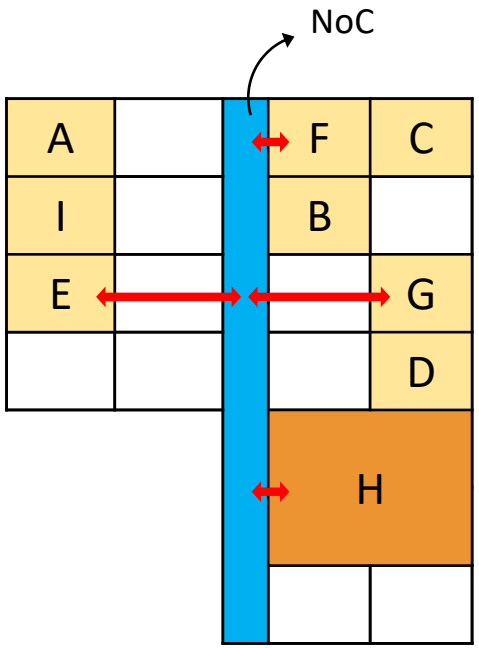
<FPGA mapping>



Evaluation

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

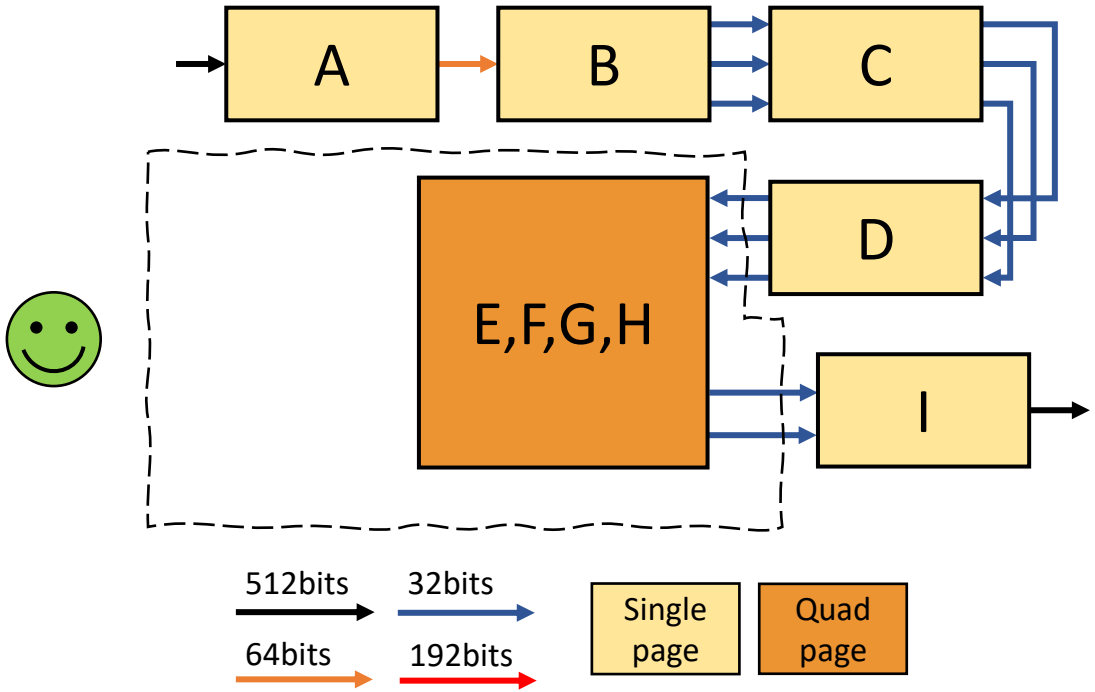


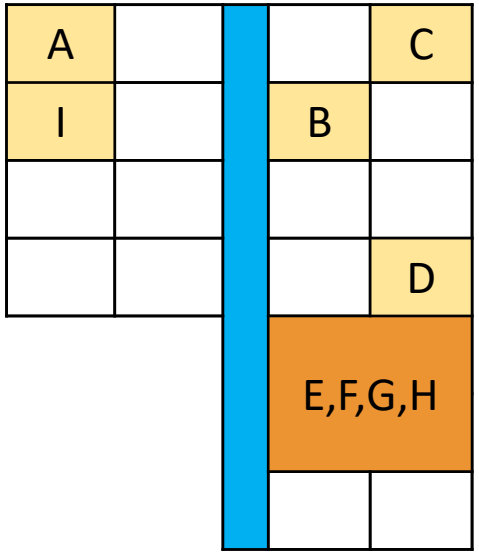


<FPGA mapping>

Evaluation

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





<FPGA mapping>

Evaluation

- Incremental development scenario with Optical Flow benchmark
  - Followed by other optimizations

<Incremental Development with our framework>

steps	Largest Page	Incremental Compile Time	App Latency
1	Quad	261s	18.7ms
2	Quad	245s	18.1ms
3	Quad	274s	16.0ms
4	Quad	288s	14.7ms
5	Single	94s	14.7ms
6	Single	107s	14.7ms
7	Quad	291s	8.7ms
8	Quad	274s	8.7ms

<Vendor tool>

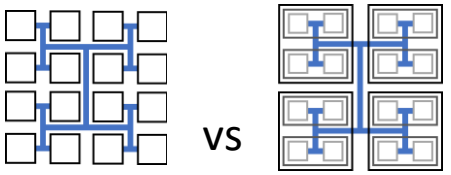
	Monolithic Compile Time	App Latency
Ver. 1	742s	19.1ms
Ver. 2	742s	19.4ms

+ Can quickly edit/compile/optimize with our framework

Evaluation

- Incremental development scenario

- Performance benefits and compilation improvement

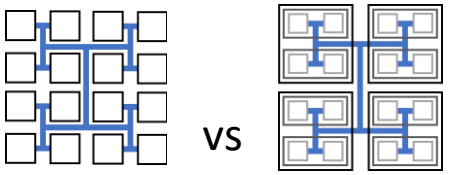


  - Demonstrate our variable-sized pages system achieve **performance improvement** compared to the fixed-sized pages system (1.4~4.9×)
  - Demonstrate our variable-sized pages system **compiles faster** than the vendor tool (2.3~5.5×)

Evaluation

- Incremental development scenario

- 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 (2.3~5.5×)

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×

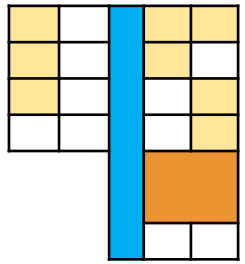
Par = Parallelization Factor

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

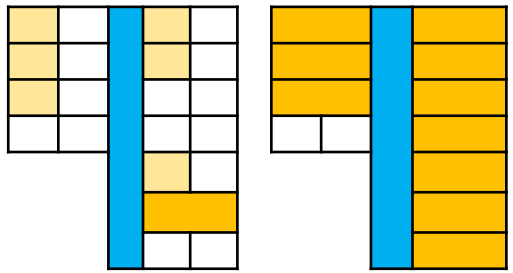
1) **Removing NoC bandwidth bottleneck by merging ops**
2) Use more area for single operator(double, quad page)

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

- If you have a friend who's complaining that it takes a long time to build hardware, please point our research to him/her
- Git repo: github.com/icgrp/prflow_nested_dfx
- Hope to hear any feedback!