Talk at ~~Intel PSG~~, San Jose
Altera!

Software-like Incremental Refinement on FPGA using Partial Reconfiguration

Dongjoon(DJ) Park
Advisor: Prof. André DeHon

Implementation of Computation Group
University of Pennsylvania

Table of Contents

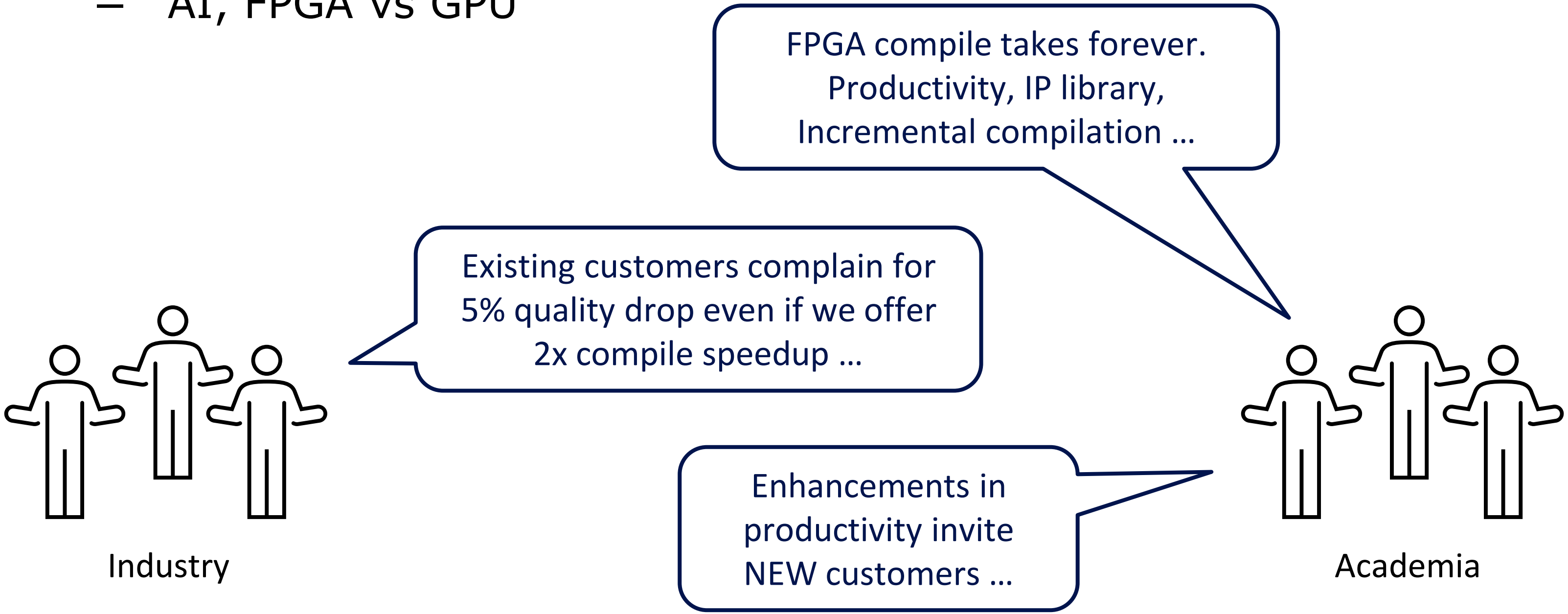
- Motivation
- Idea – Separate compilation in Parallel using Partial Reconfiguration
- Idea – More Flexibility using Hierarchical PR
- Idea – Incremental Refinement Strategy and Profiling
- Discussion & Conclusion

Table of Contents

- Motivation
- Idea – Separate compilation in Parallel using Partial Reconfiguration
- Idea – More Flexibility using Hierarchical PR
- Idea – Incremental Refinement Strategy and Profiling
- Discussion & Conclusion

Motivation

- Two days ago… Banquet at FPGA2024
  - AI, FPGA vs GPU



FPGA compile takes forever. Productivity, IP library, Incremental compilation …

Existing customers complain for 5% quality drop even if we offer 2x compile speedup …

Enhancements in productivity invite NEW customers …

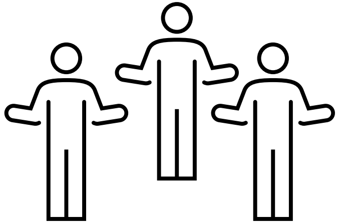
Motivation

- Two days ago… Banquet at FPGA2024
  - AI, FPGA vs GPU

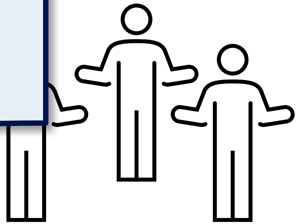
FPGA compile takes forever. Productivity, IP library, Incremental compilation …

FPGA development is much more challenging than SW development!

Enhancements in productivity invite NEW customers …



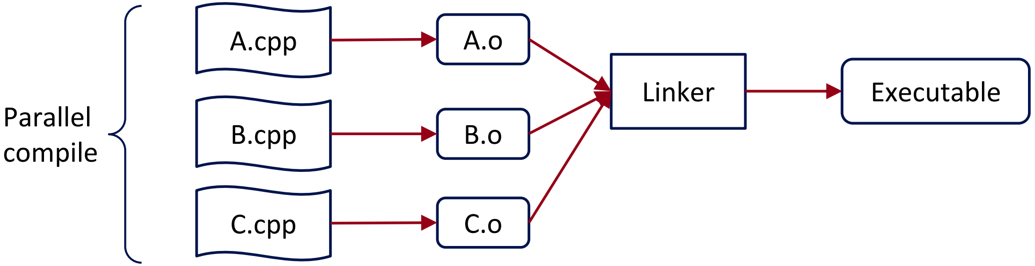
Industry



Academia

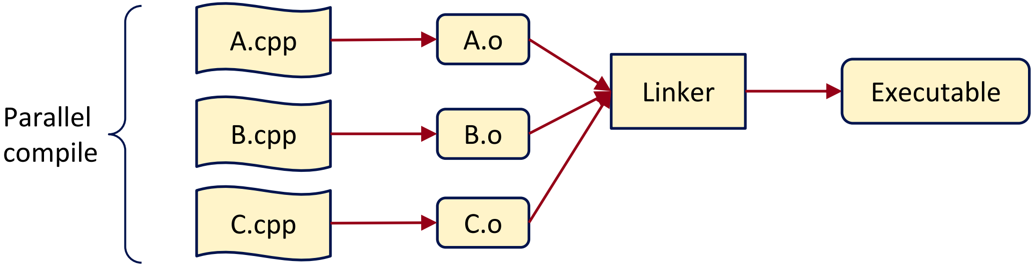
Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement



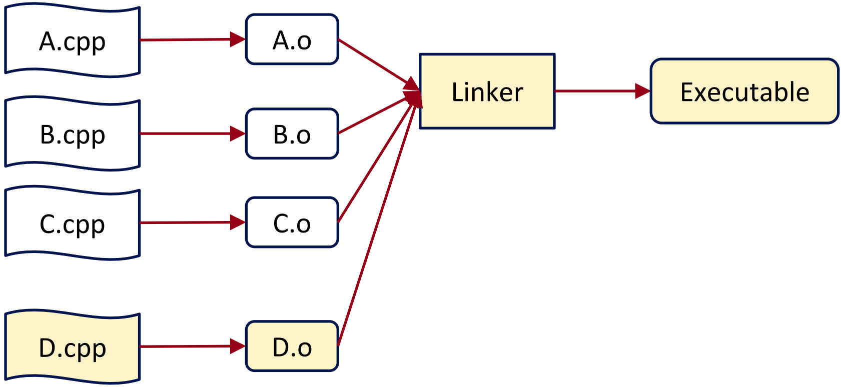
Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement



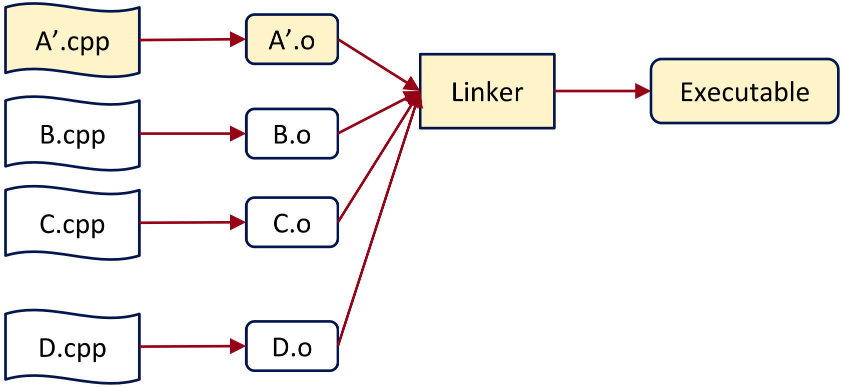
Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement



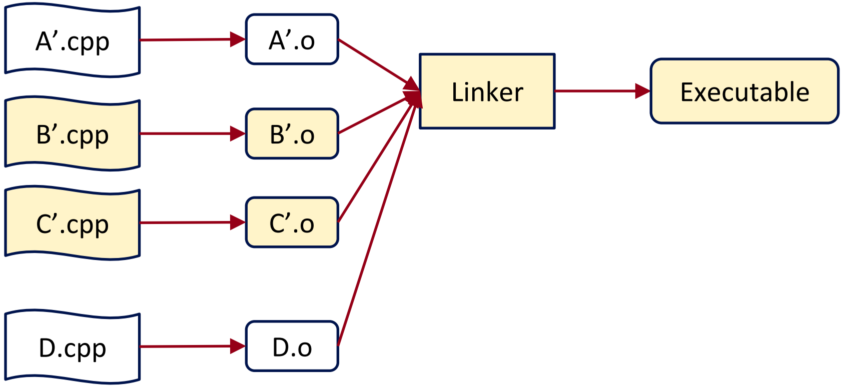
Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement



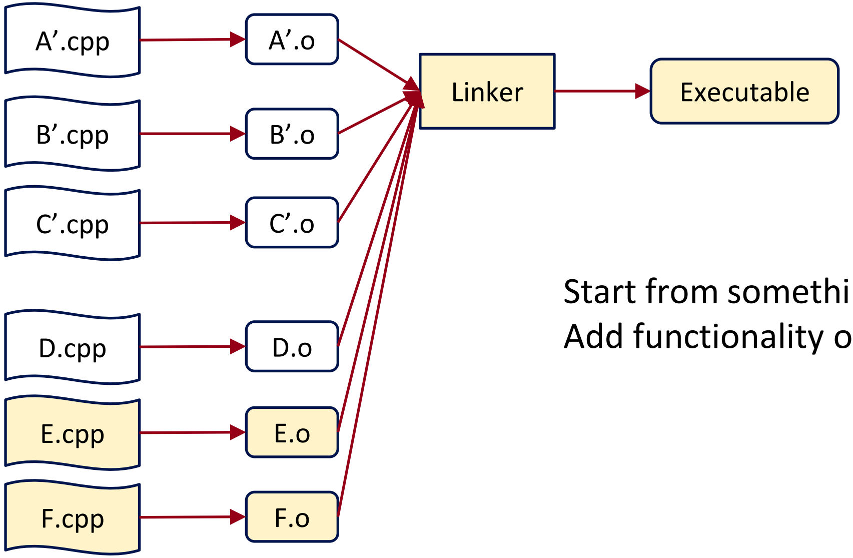
Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement



Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement

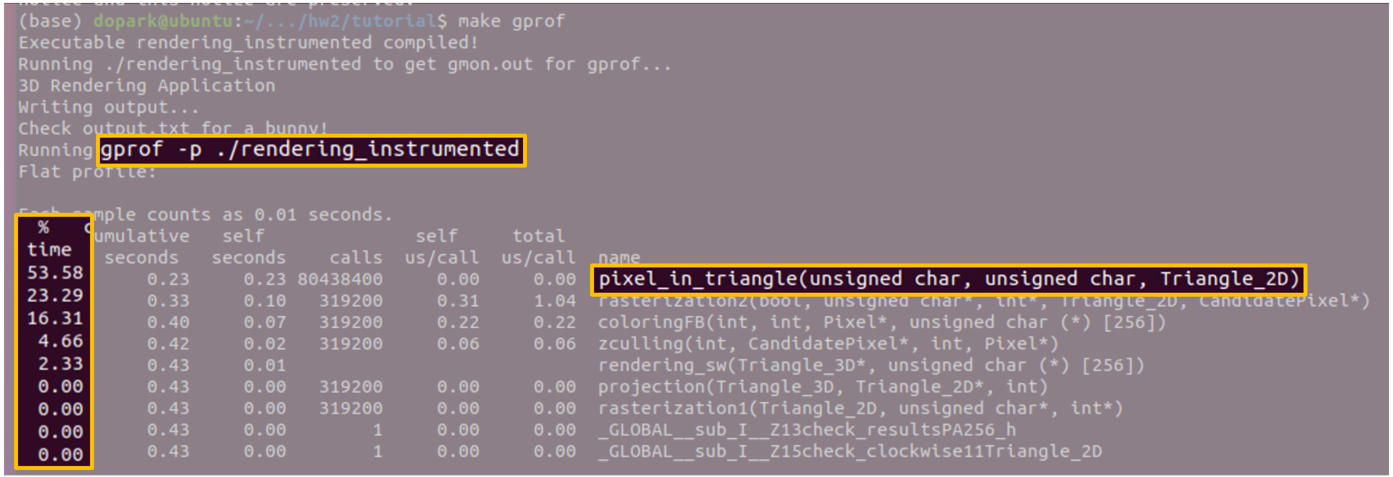


Start from something barely functional…
Add functionality one at a time…

Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools

SW engineers can easily profile the application
to investigate where the application spent its time on.

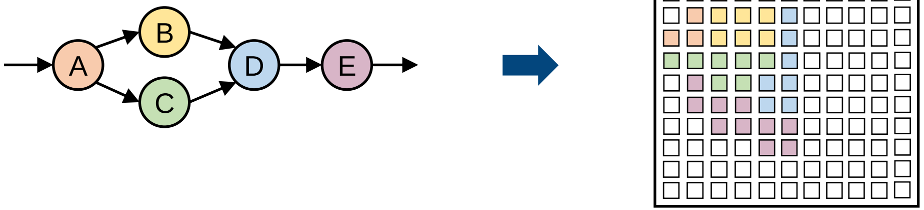


```
notice and this notice are preserved.
(base) dopark@ubuntu:~/.../hw2/tutorial$ make gprof
Executable rendering_instrumented compiled!
Running ./rendering_instrumented to get gmon.out for gprof...
3D Rendering Application
Writing output...
Check output.txt for a bunny!
Running gprof -p ./rendering_instrumented
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
53.58      0.23     0.23 80438400     0.00     0.00  pixel_in_triangle(unsigned char, unsigned char, Triangle_2D)
23.29      0.33     0.10   319200     0.31     1.04  rasterization2(bool, unsigned char*, int*, Triangle_2D, CandidatePixel*)
16.31      0.40     0.07   319200     0.22     0.22  coloringFB(int, int, Pixel*, unsigned char (*) [256])
 4.66      0.42     0.02   319200     0.06     0.06  zculling(int, CandidatePixel*, int, Pixel*)
 2.33      0.43     0.01                             rendering_sw(Triangle_3D*, unsigned char (*) [256])
 0.00      0.43     0.00   319200     0.00     0.00  projection(Triangle_3D, Triangle_2D*, int)
 0.00      0.43     0.00   319200     0.00     0.00  rasterization1(Triangle_2D, unsigned char*, int*)
 0.00      0.43     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z13check_resultsPA256_h
 0.00      0.43     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z15check_clockwise11Triangle_2D
```

Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?

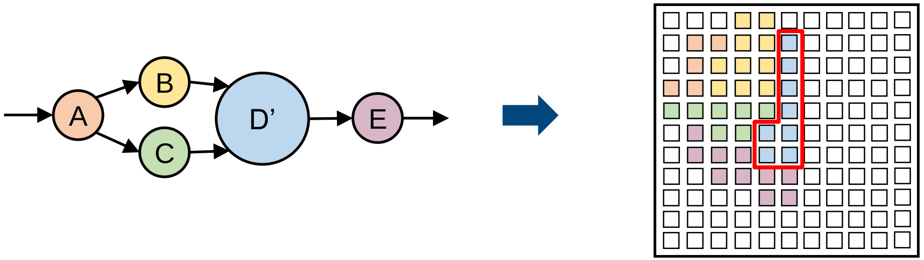


Q. Can we compile each function in parallel?
(not synthesis but place/route/bit-gen)

A. No, a design is *monolithically* compiled
➔Tool tries to optimize the entire design
➔Long compile time

Motivation

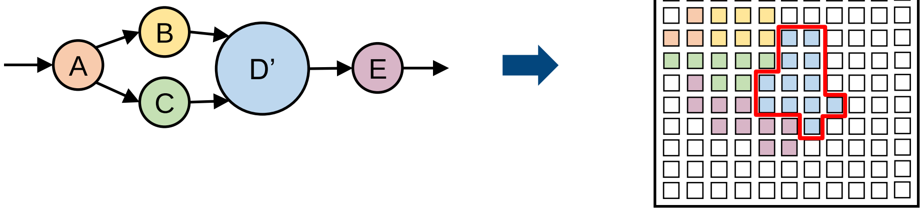
- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How’s current HW development?
  1) Parallel compile? Incremental Refinement?



Q. Can we recompile only the changed part?

Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How’s current HW development?
  1) Parallel compile? Incremental Refinement?

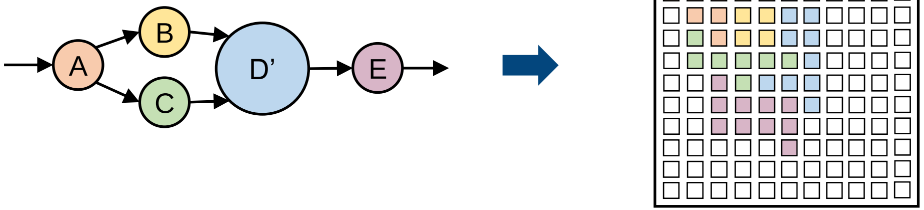


Q. Can we recompile only the changed part?

Something like this!

Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?

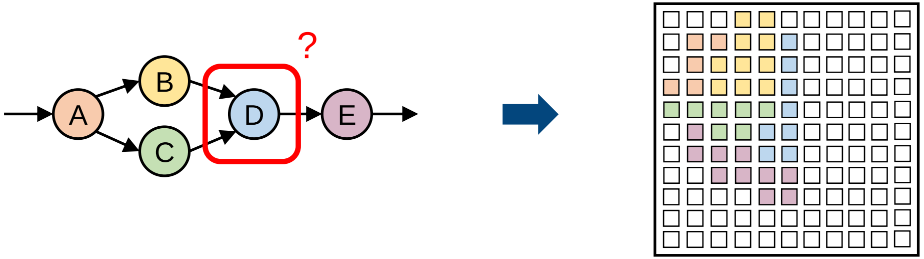


Q. Can we recompile only the changed part?

A. No, the entire design is monolithically recompiled
➔ Long compile time

Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?
  2) Profiling? Bottleneck identification?



Q. How do we know which module to refine next?

A. It's difficult to identify the bottleneck
➔ Lack of visibility on the inner state of the HW design

Motivation

- So, what is so good about SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?
  2) Profiling? Bottleneck identification?

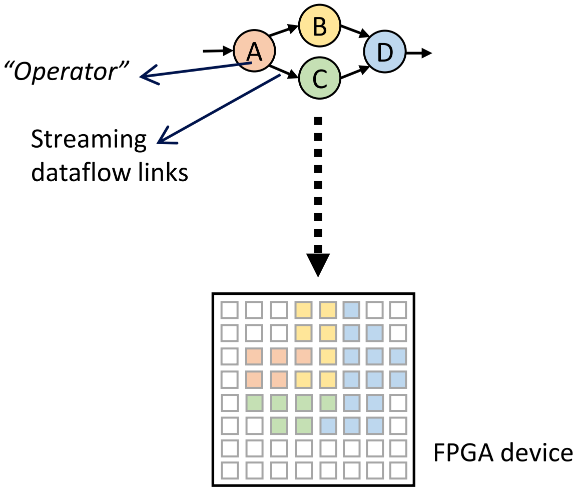
- Overall goal: SW-like FPGA design development
  - Fast Separate Compilation in Parallel using NoC + (Hierarchical) Partial Reconfiguration
  - Incremental Refinement strategy
  - Profiling using FIFO counters

Table of Contents

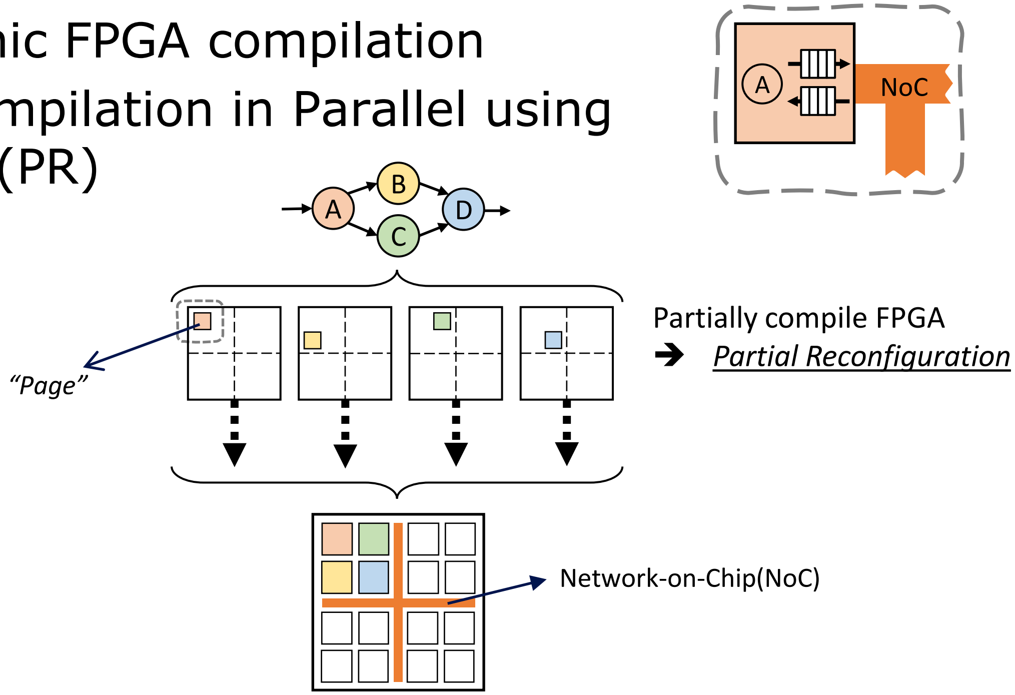
- Motivation
- Idea – Separate compilation in Parallel using Partial Reconfiguration
- Idea – More Flexibility using Hierarchical PR
- Idea – Incremental Refinement Strategy and Profiling
- Discussion & Conclusion

Idea – Separate compilation in Parallel using Partial Reconfiguration

- Problem: Slow monolithic FPGA compilation
- Idea: Fast Separate Compilation in Parallel using Partial Reconfiguration (PR)



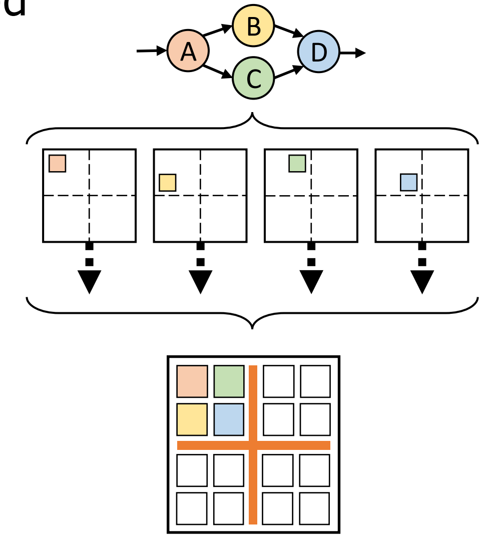
Vendor tool(Vivado, Quartus)'s slow monolithic compilation



Fast separate compilation in parallel using NoC + PR

Idea – Separate compilation in Parallel using Partial Reconfiguration

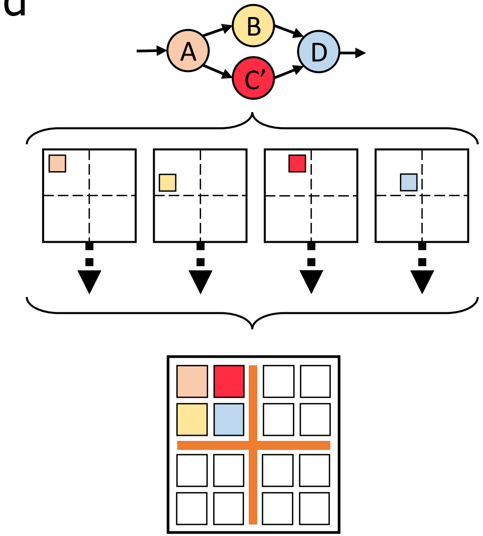
- Idea: Fast Separate Compilation in Parallel using Partial Reconfiguration (PR)
  - Pioneering work on separate compilation on FPGA using PR[1,2]
  - Parallel/Incremental compilation is supported
  - Utilized a (deflection-routed) Butterfly Fat Tree Network for the NoC



[1] Park et al., "Case for Fast FPGA Compilation Using Partial Reconfiguration", FPL 2018
[2] Xiao et al., "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks", FPT 2019

Idea – Separate compilation in Parallel using Partial Reconfiguration

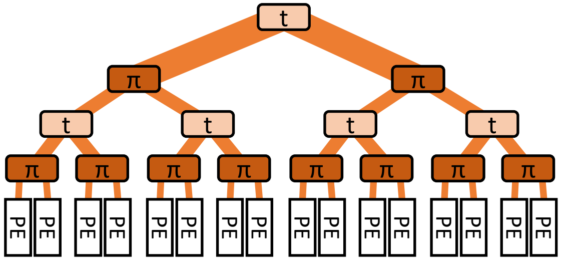
- Idea: Fast Separate Compilation in Parallel using Partial Reconfiguration (PR)
  - Pioneering work on separate compilation on FPGA using PR[1,2]
  - Parallel/Incremental compilation is supported
  - Utilized a (deflection-routed) Butterfly Fat Tree Network for the NoC



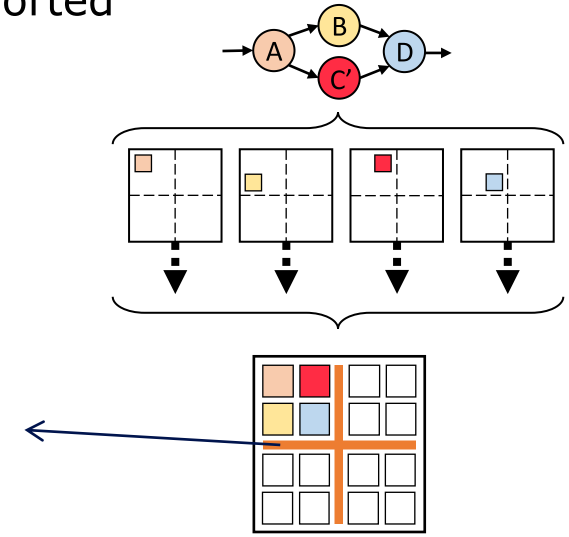
[1] Park et al., "Case for Fast FPGA Compilation Using Partial Reconfiguration", FPL 2018
[2] Xiao et al., "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks", FPT 2019

Idea – Separate compilation in Parallel using Partial Reconfiguration

- Idea: Fast Separate Compilation in Parallel using Partial Reconfiguration (PR)
  - Pioneering work on separate compilation on FPGA using PR[1,2]
  - Parallel/Incremental compilation is supported
  - Utilized a (deflection-routed) Butterfly Fat Tree Network for the NoC



<Butterfly Fat Tree, 16 PEs>



[1] Park et al., "Case for Fast FPGA Compilation Using Partial Reconfiguration", FPL 2018

[2] Xiao et al., "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks", FPT 2019

Idea – Separate compilation in Parallel using Partial Reconfiguration

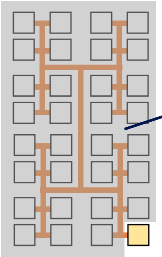
- Results
  - Demonstrated 30 min of PnR/bit-gen time with the vendor tool can be reduced to 7 min with separate compile on 31-multicore design[1]
  - More HLS benchmarks illustrated in [2] led by Yuanlong Xiao
  - Analyzed the vendor tool's compile time[2]
    - Full benefit is not achieved in [1,2] because of tool limitation
    - Even though the *static logic* is static, the vendor tool still spends time loading the design

[1] Park et al., "Case for Fast FPGA Compilation Using Partial Reconfiguration", FPL 2018

[2] Xiao et al., "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks", FPT 2019

Idea – Separate compilation in Parallel using Partial Reconfiguration

- Results
  - Demonstrated 30 min of PnR/bit-gen time with the vendor tool can be reduced to 7 min with separate compile on 31-multicore design[1]
  - More HLS benchmarks illustrated in [2] led by Yuanlong Xiao
  - Analyzed the vend
    - Full benefit is
    - Even though t spends time lo



- This part is static(fixed), so ideally, we don't want to spend any time compiling.

➔ But Vivado does spend time even for the fixed static logic.

- Larger static design leads to longer compile time in PR[2]
- Recently observed the same behavior on Quartus PR

Static Logic and Compile Time

[1] Park et al., "Case for Fast FPGA Compilation Using Partial Reconfiguration", FPL 2018

[2] Xiao et al., "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks", FPT 2019

Idea – Separate compilation in Parallel using Partial Reconfiguration

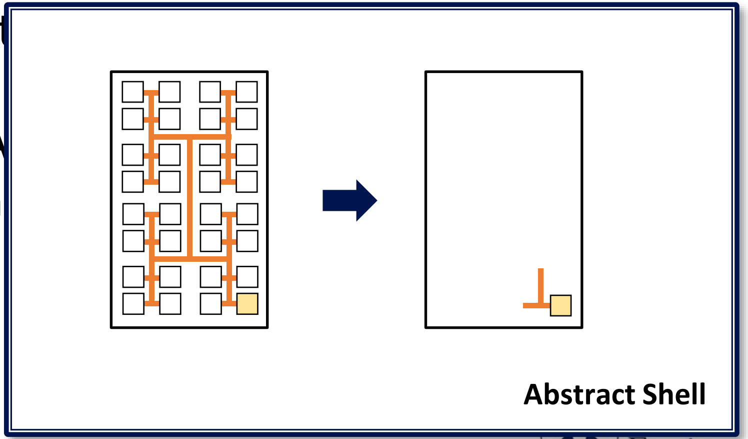
- Results
  - Demonstrated 30 min of PnR/bit-gen time with the vendor tool can be reduced to 7 min with separate compile on 31-multicore design[1]
  - More HLS benchmarks illustrated in [2] led by Yuanlong Xiao
  - Analyzed the vendor tool’s compile time[2]
    - Full benefit is not achieved in [1,2] because of tool limitation
    - Even though the *static logic* is static, the vendor tool still spends time loading the design
    - This issue was mitigated with “Abstract Shell” from Xilinx
      - contains minimal logical and physical database

[1] Park et al., “Case for Fast FPGA Compilation Using Partial Reconfiguration”, FPL 2018

[2] Xiao et al., “Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks”, FPT 2019

Idea – Separate compilation in Parallel using Partial Reconfiguration

- Results
  - Demonstrated 30 min of PnR/bit-gen time with the vendor tool can be reduced to 7 min with separate compile on 31-multicore design[1]
  - More HLS benchmarks illustrated in [2] led by Yuanlong Xiao
  - Analyzed the vendor tool’s compile time[2]
    - Full benefit is not achieved in [1,2] because of tool limitation
    - Even though the *static logic* is st spends time loading the design
    - This issue was mitigated with “A
      - contains minimal logical and ph



[1] Park et al., “Case for Fast FPGA Compilation Using Partial Reconfiguration”, FPL 2018
[2] Xiao et al., “Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks”, FPT 2019

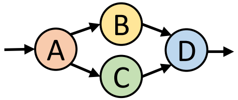
Idea – Separate compilation in Parallel using Partial Reconfiguration

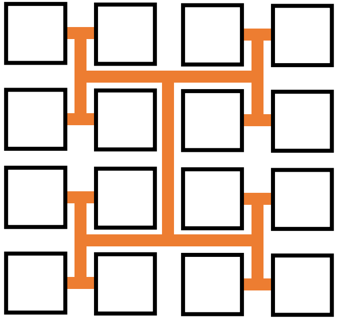
- Results
  - Demonstrated 30 min of PnR/bit-gen time with the vendor tool can be reduced to 7 min with separate compile on 31-multicore design[1]
  - More HLS benchmarks illustrated in [2] led by Yuanlong Xiao
  - Analyzed the vendor tool's compile time[2]
    - Full benefit is not achieved in [1,2] because of tool limitation
    - Even though the *static logic* is static, the vendor tool still spends time loading the design
    - This issue was mitigated with "Abstract Shell" from Xilinx
      - contains minimal logical and physical database
    - Intel Quartus has "Fast Preservation"
      - simplifies the logic of a preserved partition during compilation to only the interface logic between the partition boundary and the rest of the design

[1] Park et al., "Case for Fast FPGA Compilation Using Partial Reconfiguration", FPL 2018
[2] Xiao et al., "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks", FPT 2019

Idea – Separate compilation in Parallel using Partial Reconfiguration

- Q. Does the user have to decompose a design into regularly-sized operators?





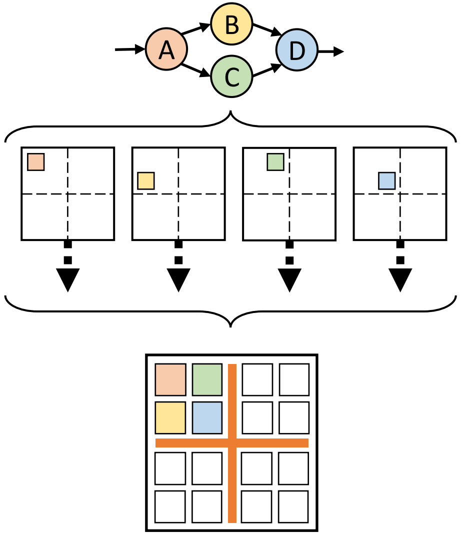
<Fixed-sized pages>

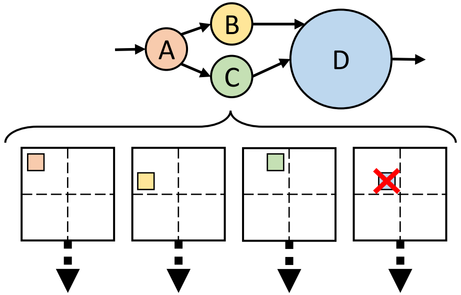
Table of Contents

- Motivation
- Idea – Separate compilation in Parallel using Partial Reconfiguration
- Idea – More Flexibility using Hierarchical PR
- Idea – Incremental Refinement Strategy and Profiling
- Discussion & Conclusion

Idea – More Flexibility using Hierarchical PR

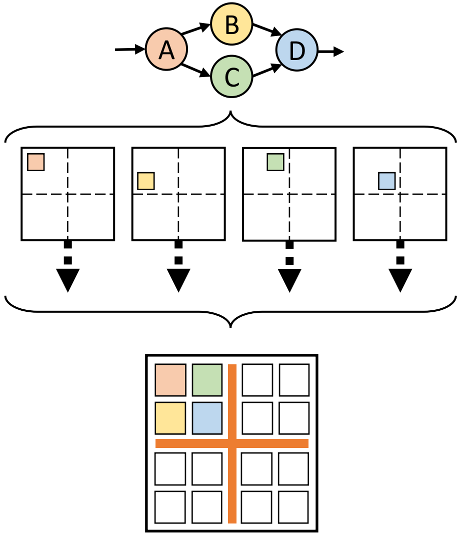
- Problem: Fixed-sized pages in separate compilations approaches
  - What if the sizes of operators are unbalanced?

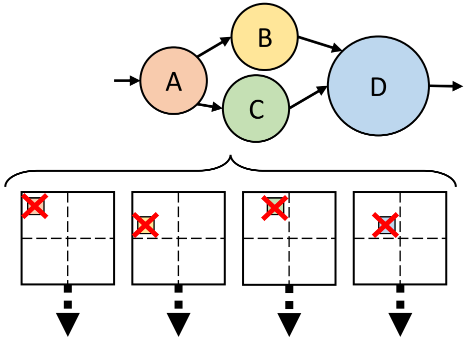




Idea – More Flexibility using Hierarchical PR

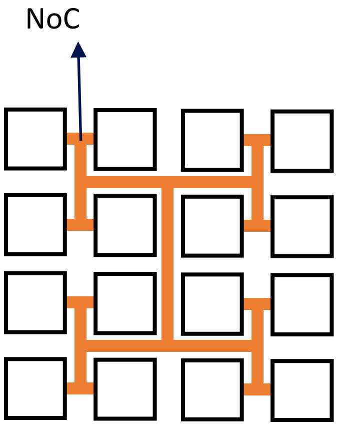
- Problem: Fixed-sized pages in separate compilations approaches
  - What if the sizes of operators are unbalanced?
  - What if a user wants to optimized further?



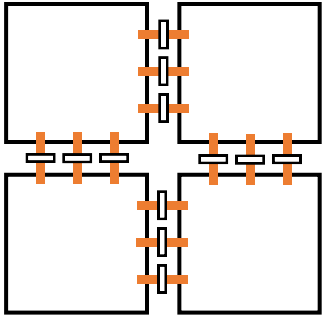


Idea – More Flexibility using Hierarchical PR

- Problem: Fixed-sized pages in separate compilations approaches
  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.



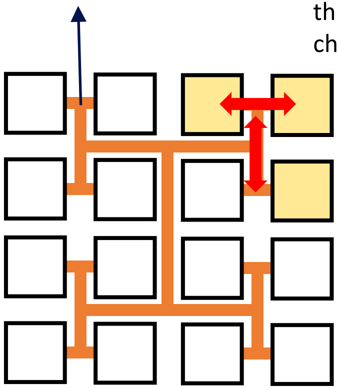
Small Pages



Large Pages

Idea – More Flexibility using Hierarchical PR

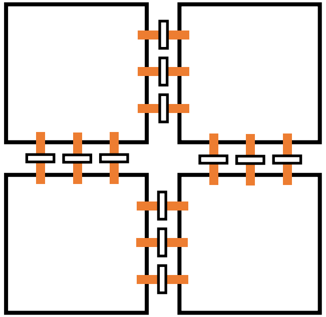
- Problem: Fixed-sized pages in separate compilations approaches
  1) If the pages are large, it reduces the benefit of separate compilations.
  2) If the pages are small, the users need to manually divide the design into small operators. Also causes NoC bandwidth bottleneck.



NoC

Too much data going through limited channels!

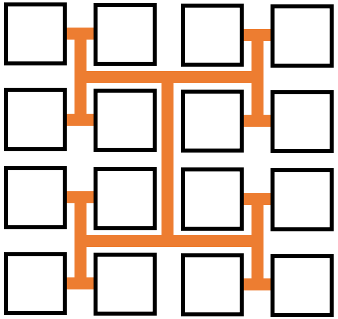
Small Pages



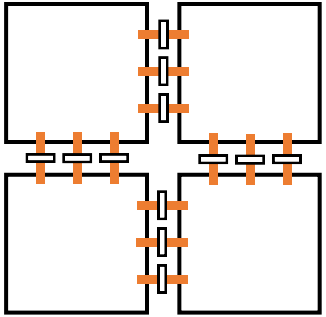
Large Pages

Idea – More Flexibility using Hierarchical PR

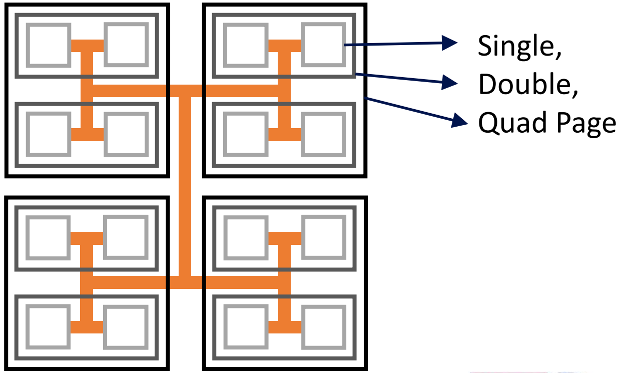
- Idea: Flexible-sized PR pages using Hierarchical PR[3]
  - Supported by Xilinx since tool ver. 2020.1 (2020)
    - Also available in Quartus
  - Partial region inside partial region



Small Pages



Large Pages

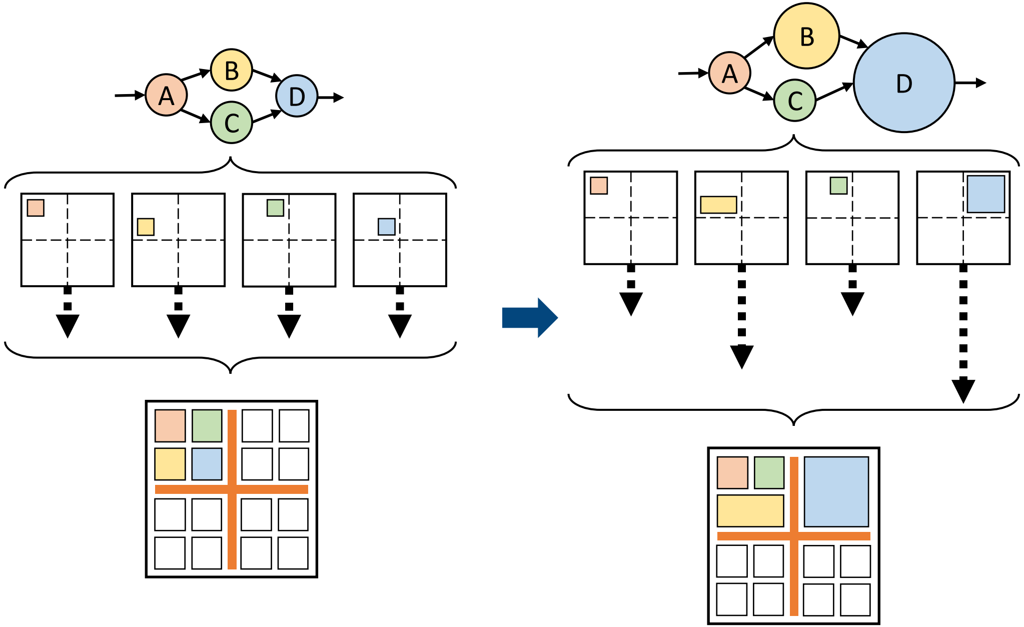


Hierarchical Pages [3]

[3] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Idea – More Flexibility using Hierarchical PR

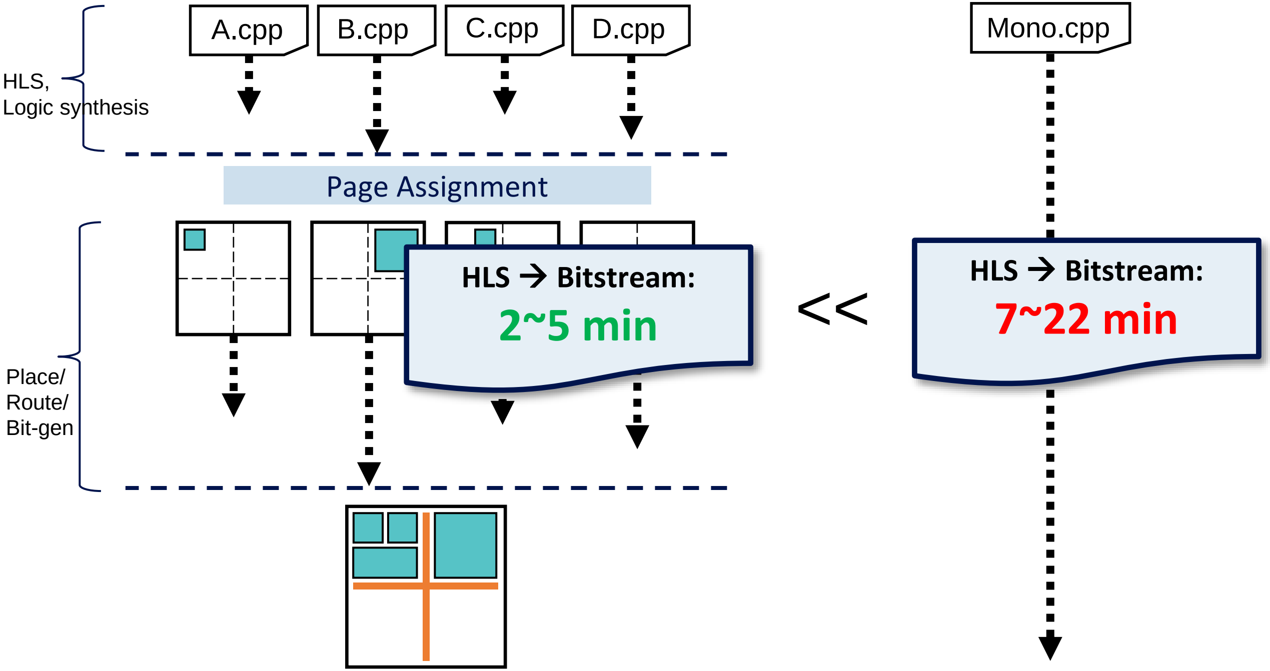
- Idea: Flexible-sized PR pages using Hierarchical PR[3]



[3] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Idea – More Flexibility using Hierarchical PR

- Idea: Flexible-sized PR pages using Hierarchical PR[3]



HLS → Bitstream: 2~5 min

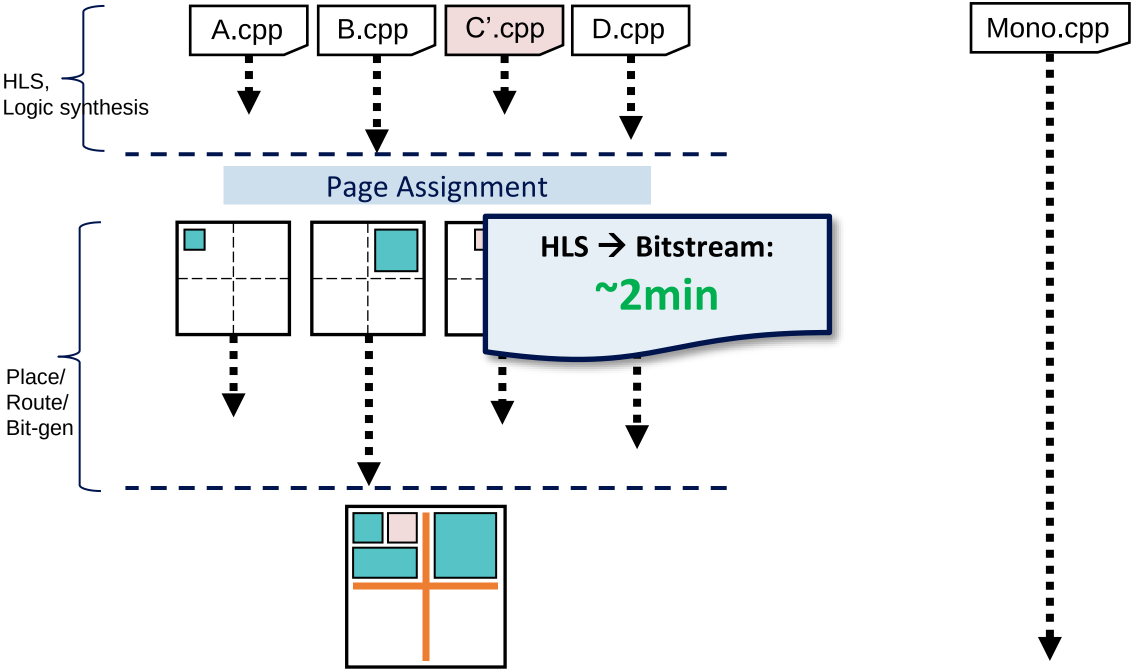
<<

HLS → Bitstream: 7~22 min

[3] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Idea – More Flexibility using Hierarchical PR

- Idea: Flexible-sized PR pages using Hierarchical PR[3]



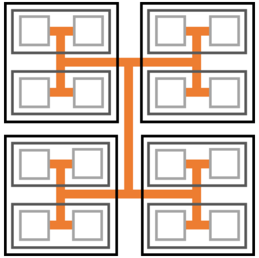
HLS → Bitstream:

~2min

[3] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Idea – More Flexibility using Hierarchical PR

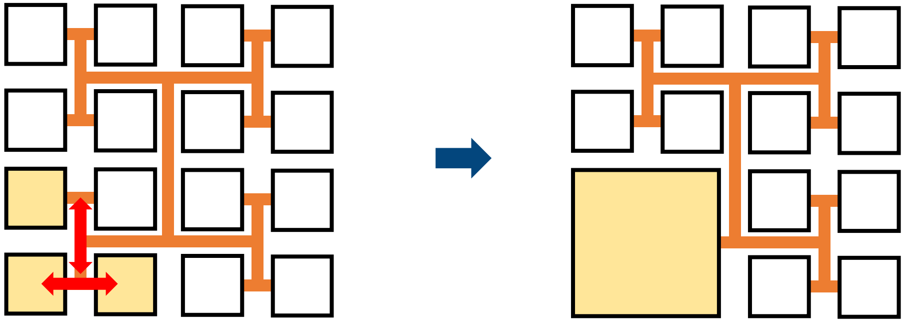
- Idea: Flexible-sized PR pages using Hierarchical PR[3]
- Advantages
  - Fine-grained separate compilations with single pages
    ➔ maximize benefits of fast separate compilations
  - Users are not forced to decompose a design into small operators. They can use double pages or quad pages.
    ➔ flexible framework
  - Useful in incremental refinement
    ➔ Users can quickly start from natural decomposition and incrementally refine just like SW!



[3] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Idea – More Flexibility using Hierarchical PR

- Results – detailed results in [3]
  - Improves application performance by 1.4~4.9x compared to a fixed-sized pages system on Rosetta HLS benchmarks[4]
    - Remove NoC bandwidth by merging ops
    - Use more area for single operator



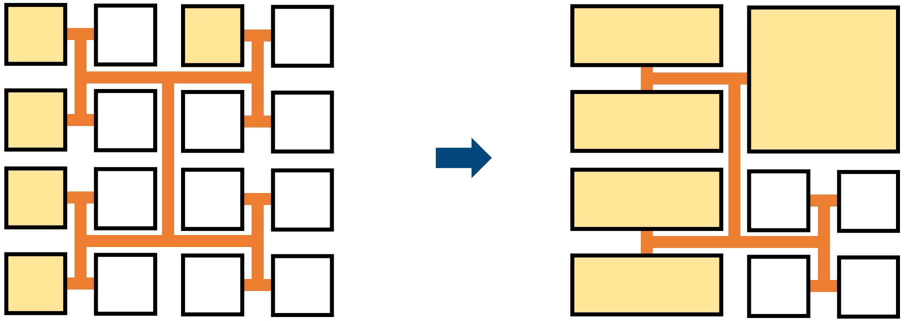
<Remove NoC bandwidth bottleneck by merging ops>

[3] Park et al., “Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration”, FPT 2022

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

Idea – More Flexibility using Hierarchical PR

- Results – detailed results in [3]
  - Improves application performance by 1.4~4.9x compared to a fixed-sized pages system on Rosetta HLS benchmarks[4]
    - Remove NoC bandwidth by merging ops
    - Use more area for single operator



<Use Double/Quad page for a single operator>

[3] Park et al., “Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration”, FPT 2022

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

Idea – More Flexibility using Hierarchical PR

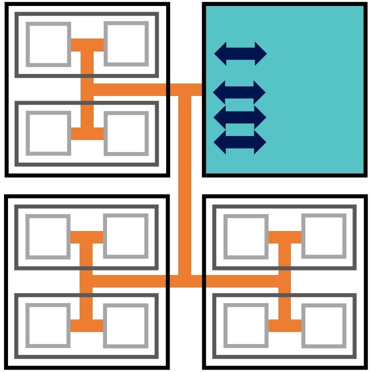
- Results – detailed results in [3]
  - Improves application performance by 1.4~4.9x compared to a fixed-sized pages system on Rosetta HLS benchmarks[4]
    - Remove NoC bandwidth by merging ops
    - Use more area for single operator
  - While compiling 2.2~5.3x faster than the vendor tool
    - In incremental refinement scenario, a single page takes less than 2 minutes to compile (HLS → partial bitstream)

[3] Park et al., “Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration”, FPT 2022

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

Idea – More Flexibility using Hierarchical PR

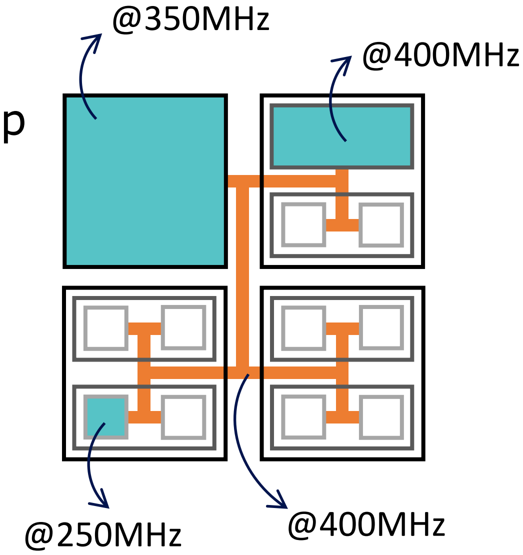
- More enhancements on the separate compilation framework[5]
  - Mitigate NoC bandwidth bottleneck
    - Use multiple NoC interfaces



[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

Idea – More Flexibility using Hierarchical PR

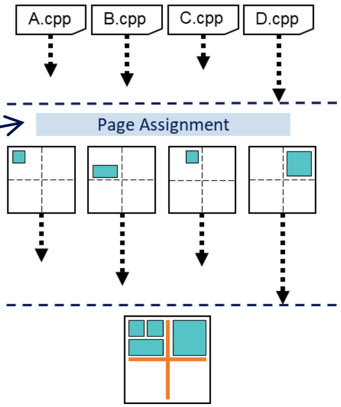
- More enhancements on the separate compilation framework[5]
  - Mitigate NoC bandwidth bottleneck
    - Use multiple NoC interfaces
  - Support for multiple clock frequencies for each op
    - NoC runs @ 400MHz
    - Operators run @ 200~400MHz

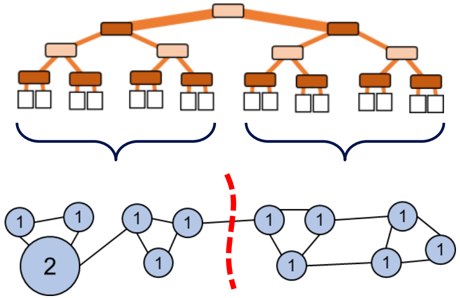


[5] Park et al., “REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs”, FPGA 2024

Idea – More Flexibility using Hierarchical PR

- More enhancements on the separate compilation framework[5]
  - Mitigate NoC bandwidth bottleneck
    - Use multiple NoC interfaces
  - Support for multiple clock frequencies for each op
    - NoC runs @ 400MHz
    - Operators run @ 200~400MHz
  - Page assignment based on recursive graph bipartitioning
    - Reduce traffic over NoC

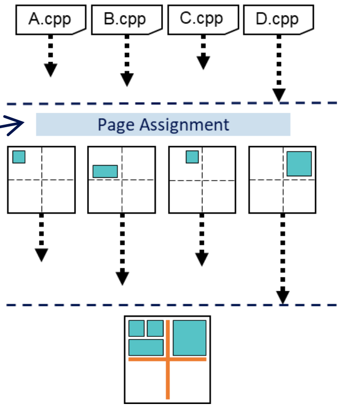


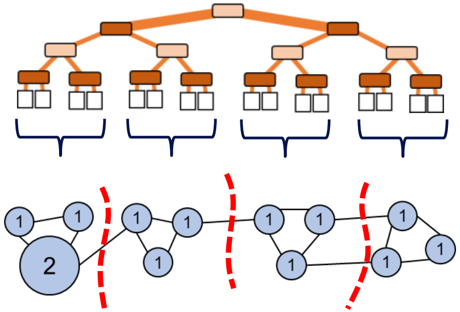


[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

Idea – More Flexibility using Hierarchical PR

- More enhancements on the separate compilation framework[5]
  - Mitigate NoC bandwidth bottleneck
    - Use multiple NoC interfaces
  - Support for multiple clock frequencies for each op
    - NoC runs @ 400MHz
    - Operators run @ 200~400MHz
  - Page assignment based on recursive graph bipartitioning
    - Reduce traffic over NoC

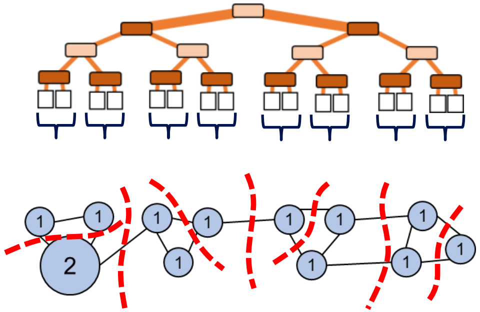


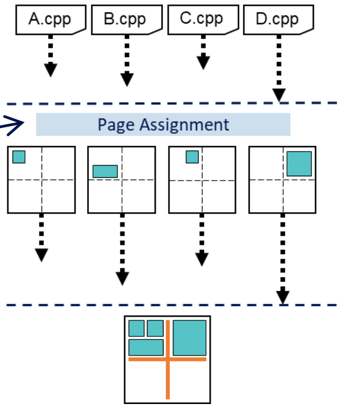


[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

Idea – More Flexibility using Hierarchical PR

- More enhancements on the separate compilation framework[5]
  - Mitigate NoC bandwidth bottleneck
    - Use multiple NoC interfaces
  - Support for multiple clock frequencies for each op
    - NoC runs @ 400MHz
    - Operators run @ 200~400MHz
  - Page assignment based on recursive graph bipartitioning
    - Reduce traffic over NoC
  - More enhancements in [5]





[5] Park et al., “REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs”, FPGA 2024

Table of Contents

- Motivation
- Idea – Separate compilation in Parallel using Partial Reconfiguration
- Idea – More Flexibility using Hierarchical PR
- Idea – Incremental Refinement Strategy and Profiling
- Discussion & Conclusion

Idea – Incremental Refinement Strategy and Profiling

- Remember, the goal: “SW-like FPGA design development”
  - Fast Separate Compilation in Parallel using NoC + (Hierarchical) Partial Reconfiguration
  - Incremental Refinement strategy
  - Profiling using FIFO counters

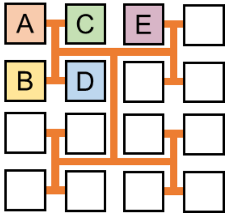
Idea – Incremental Refinement Strategy and Profiling

- Remember, the goal: “SW-like FPGA design development”
  - ~~Fast Separate Compilation in Parallel using NoC + (Hierarchical) Partial Reconfiguration~~
  - Incremental Refinement strategy
  - Profiling using FIFO counters

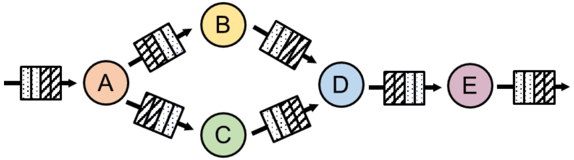
- Problem: Is the previous NoC+PR system enough for the incremental refinement on FPGA designs?

Idea – Incremental Refinement Strategy and Profiling

- Problem: Is the previous NoC+PR system enough for the incremental refinement on FPGA designs?
  - NoC-based system
    - Pro: Faster compile
      - Parallel, incremental
    - Con: NoC overhead
      - Area, Bandwidth

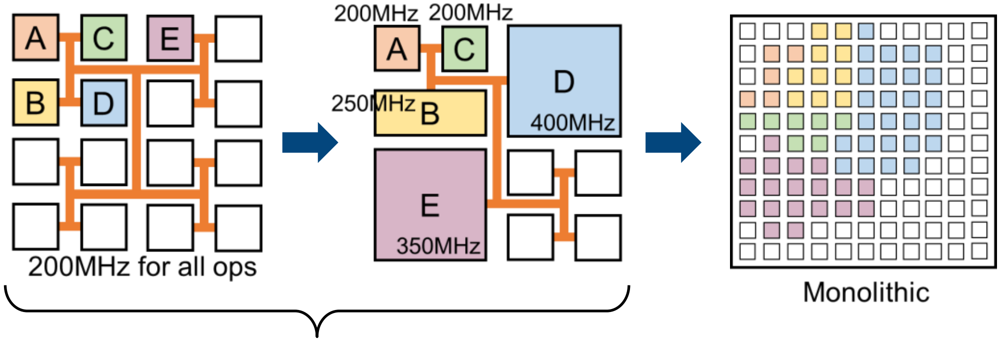


  - Monolithic system
    - Pro: No NoC overhead
    - Con: Slow compile



Idea – Incremental Refinement Strategy and Profiling

- Idea: Fast incremental refinement strategy[5]
  - Start with the **NoC-based** system
  - **Identify the bottleneck** and select the next design point
  - When a design can't be improved in the NoC-based system, (e.g. not enough area in PR page, design space is all explored) migrate to the **monolithic** system
  - **Continue** to identify the bottleneck and select the next design point



Fast compile with NoC + PR

NoC is removed!

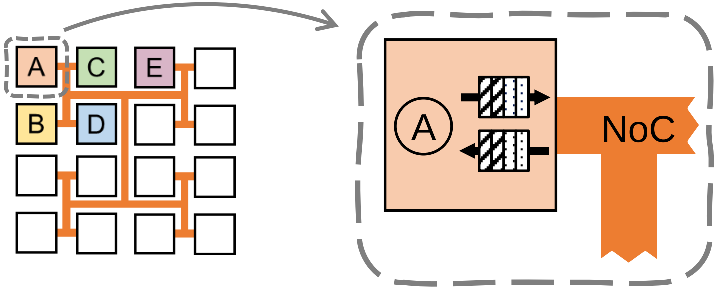
[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

Idea – Incremental Refinement Strategy and Profiling

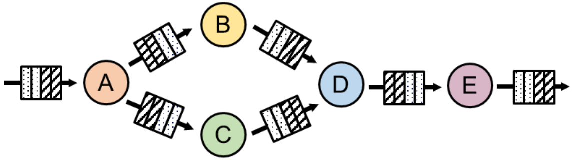
- Problem: No profiling capability. How to identify a bottleneck of a design in HW?
- Idea: Bottleneck identification using FIFO counters

Recall!

  - NoC-based system
    - Pro: Faster compile
      - Parallel, incremental
    - Con: NoC overhead
      - Area, Bandwidth

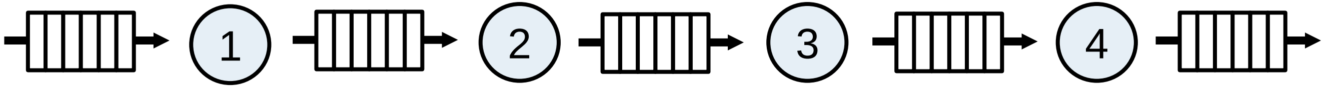


  - Monolithic system
    - Pro: No NoC overhead
    - Con: Slow compile



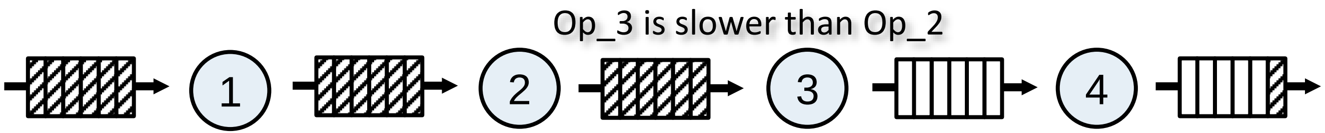
Idea – Incremental Refinement Strategy and Profiling

- Idea: Bottleneck identification using FIFO counters
  - High-level intuition



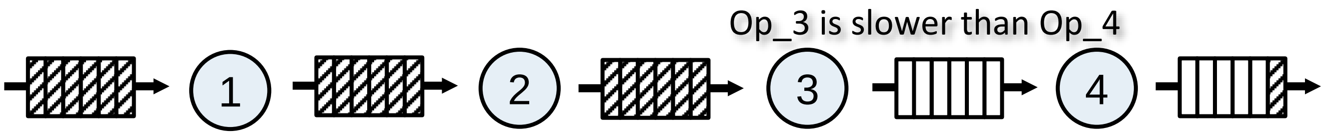
Idea – Incremental Refinement Strategy and Profiling

- Idea: Bottleneck identification using FIFO counters
  - High-level intuition



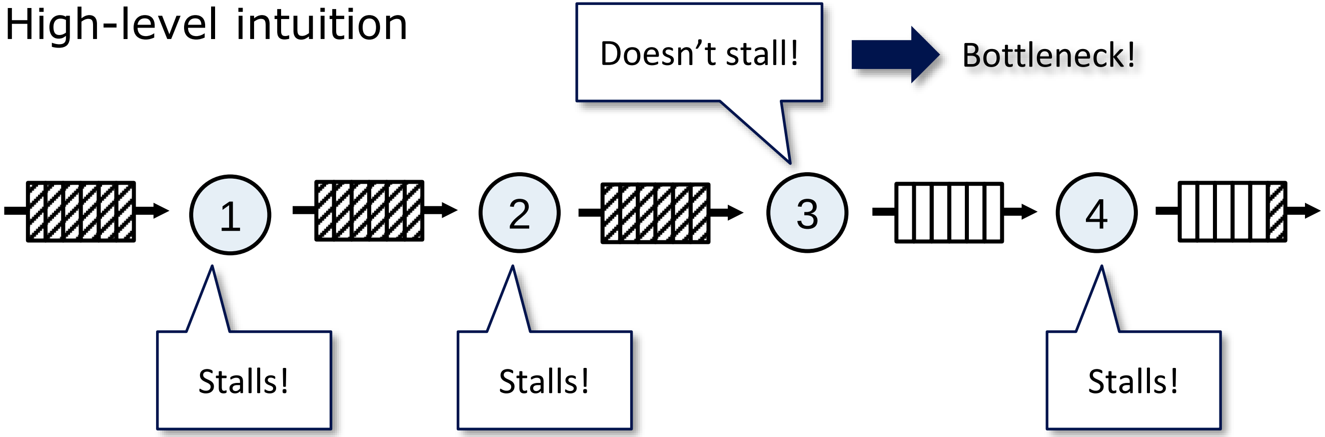
Idea – Incremental Refinement Strategy and Profiling

- Idea: Bottleneck identification using FIFO counters
  - High-level intuition



Idea – Incremental Refinement Strategy and Profiling

- Idea: Bottleneck identification using FIFO counters
  - High-level intuition



Idea – Incremental Refinement Strategy and Profiling

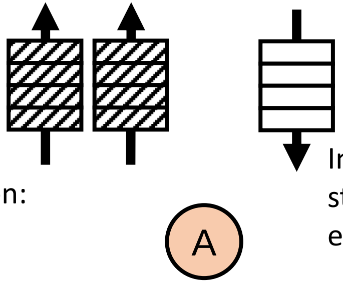
- Idea: Bottleneck identification using FIFO counters[5]

1) bottleneck operator

→ embedded in both NoC system, monolithic system

NoC (NoC system) or
Other ops. (Monolithic system)

Count the *stalls*!



Output FIFO
stall condition:
full && valid

Input FIFO
stall condition:
empty && ready

Stall condition: at least one FIFO stalls, stall cnt++

➔ Op with the least stall cnts may be the bottleneck

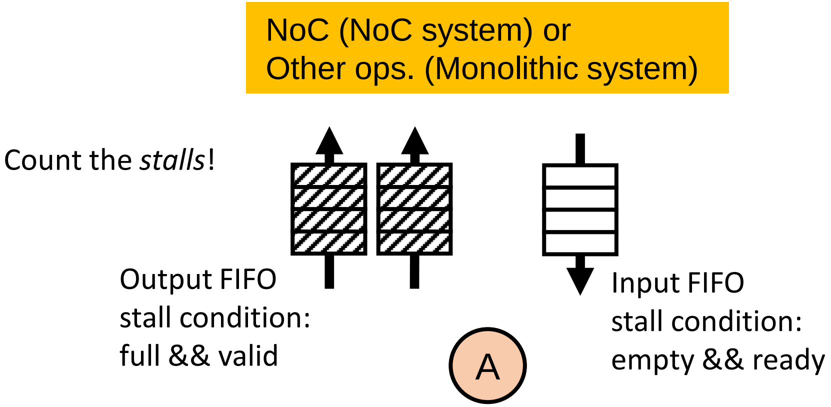
[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

Idea – Incremental Refinement Strategy and Profiling

- Idea: Bottleneck identification using FIFO counters[5]

1) bottleneck operator

→ embedded in both NoC system, monolithic system

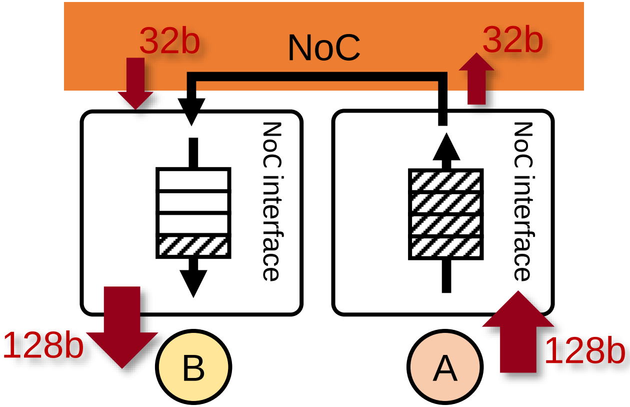


Stall condition: at least one FIFO stalls, stall cnt++

➔ Op with the least stall cnts may be the bottleneck

2) NoC bandwidth bottleneck

→ embedded in only NoC system



- Harms application performance
- Wrong bottleneck operator can be identified

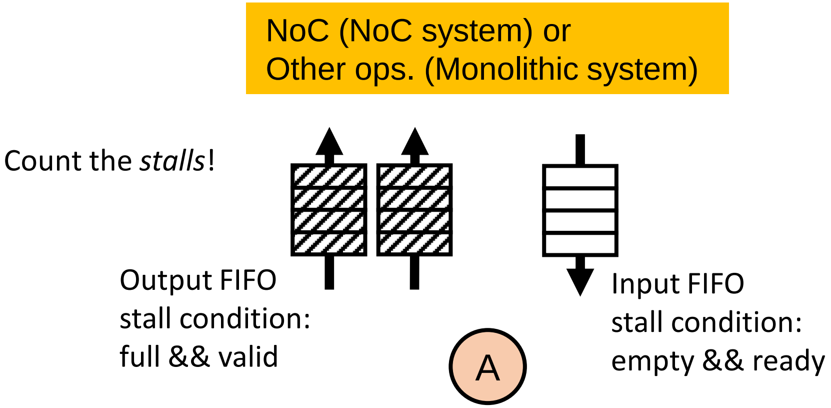
[5] Park et al., “REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs”, FPGA 2024

Idea – Incremental Refinement Strategy and Profiling

- Idea: Bottleneck identification using FIFO counters[5]

1) bottleneck operator

→ embedded in both NoC system, monolithic system



NoC (NoC system) or
Other ops. (Monolithic system)

Count the *stalls*!

Output FIFO
stall condition:
full && valid

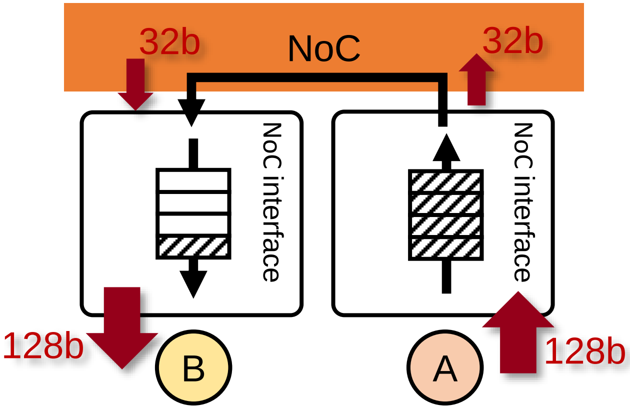
Input FIFO
stall condition:
empty && ready

Stall condition: at least one FIFO stalls, stall cnt++

➔ Op with the least stall cnts may be the bottleneck

2) NoC bandwidth bottleneck

→ embedded in only NoC system



If A's Output FIFO's full↑ && B's Input FIFO's full↓

➔ NoC bandwidth may be the bottleneck

[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

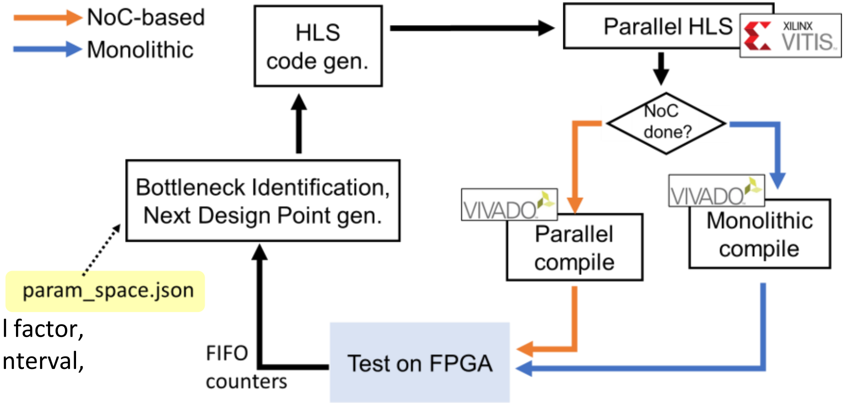
Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study
  - Observe application performance improvement with bottleneck identification
  - Compare design tuning time of our fast incremental refinement strategy vs monolithic-only flow

Idea – Incremental Refinement Strategy and Profiling

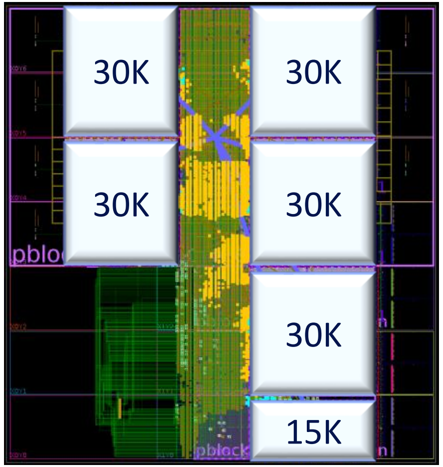
- Results: Design Space Exploration (DSE) case study

- AMD Vitis, Vitis HLS, Vivado, 2022.1
- AMD Ryzen 5950X, 16 core, 32 threads
- 128 GB RAM
- AMD ZCU102, UltraScale+ ZU9EG



e.g.: unroll factor, Initiation Interval, etc

<Automated DSE experiment overview>



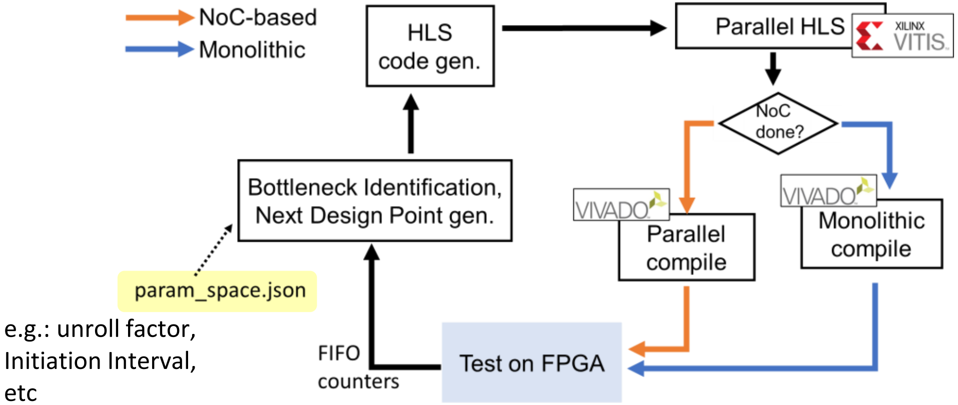
<NoC-based system overlay>

Orange: NoC
Cyan: pipeline regs (placed near PR pages)

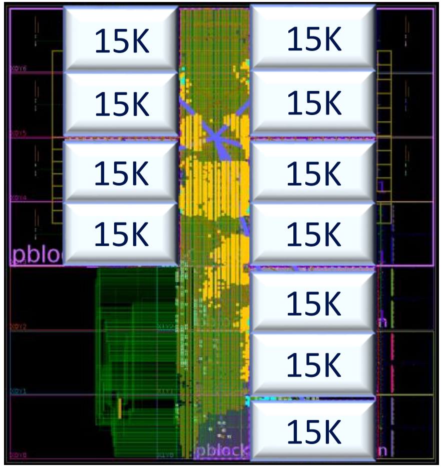
Idea – Incremental Refinement Strategy and Profiling

- AMD Vitis, Vitis HLS, Vivado, 2022.1
- AMD Ryzen 5950X, 16 core, 32 threads
- 128 GB RAM
- AMD ZCU102, UltraScale+ ZU9EG

- Results: Design Space Exploration (DSE) case study



<Automated DSE experiment overview>



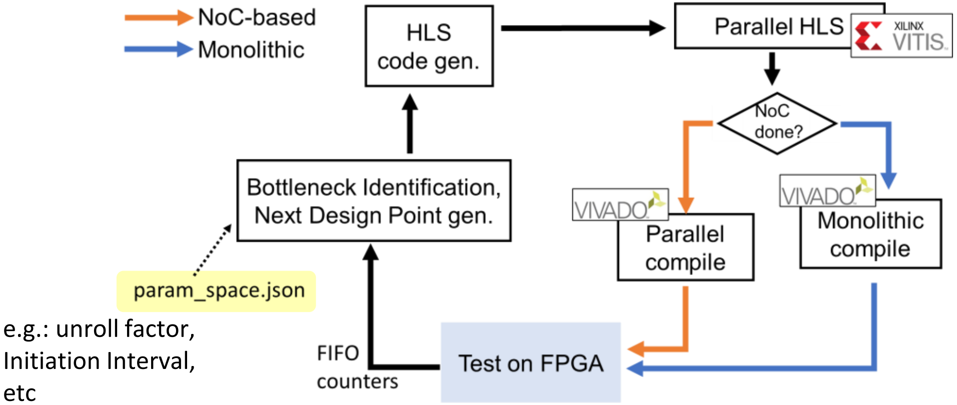
<NoC-based system overlay>

Orange: NoC
Cyan: pipeline regs (placed near PR pages)

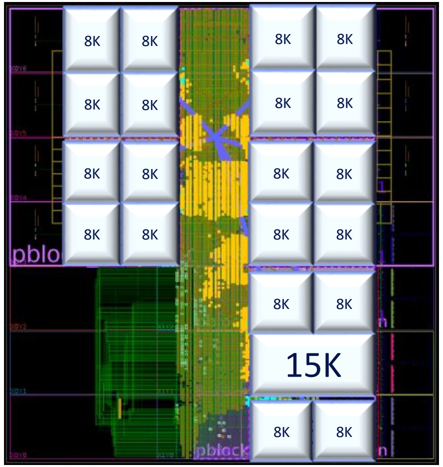
Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study

- AMD Vitis, Vitis HLS, Vivado, 2022.1
- AMD Ryzen 5950X, 16 core, 32 threads
- 128 GB RAM
- AMD ZCU102, UltraScale+ ZU9EG



<Automated DSE experiment overview>



<NoC-based system overlay>

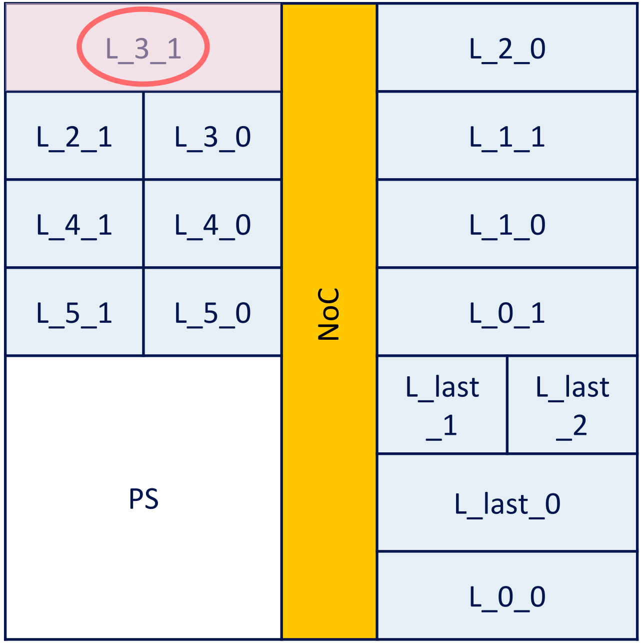
Orange: NoC
Cyan: pipeline regs (placed near PR pages)

Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study

Example: CNN-2 benchmark

PE=8 → PE=16

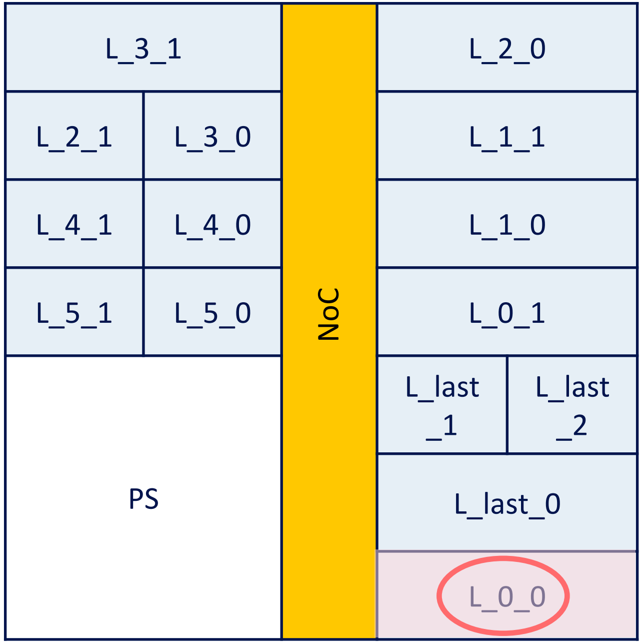


<NoC-based system>

Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study

Example: CNN-2 benchmark



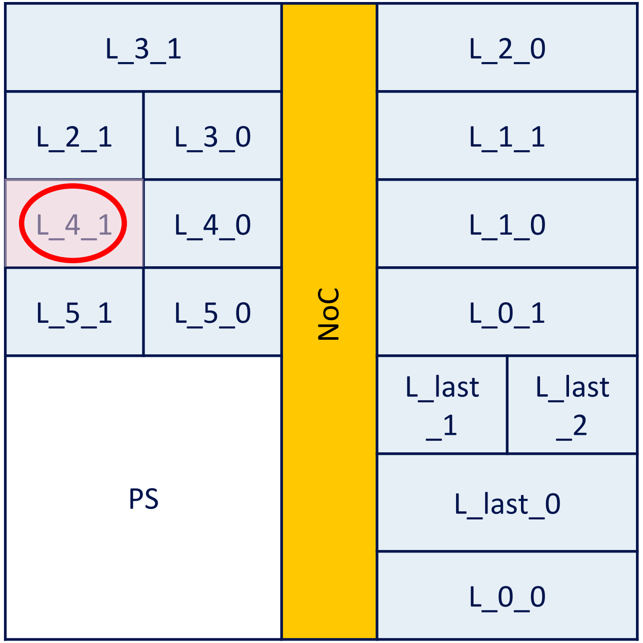
200MHz → 250MHz

<NoC-based system>

Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study

Example: CNN-2 benchmark



L_3_1

L_2_0

L_2_1

L_3_0

L_1_1

L_4_1

L_4_0

L_1_0

L_5_1

L_5_0

L_0_1

NoC

L_last_1

L_last_2

PS

L_last_0

L_0_0

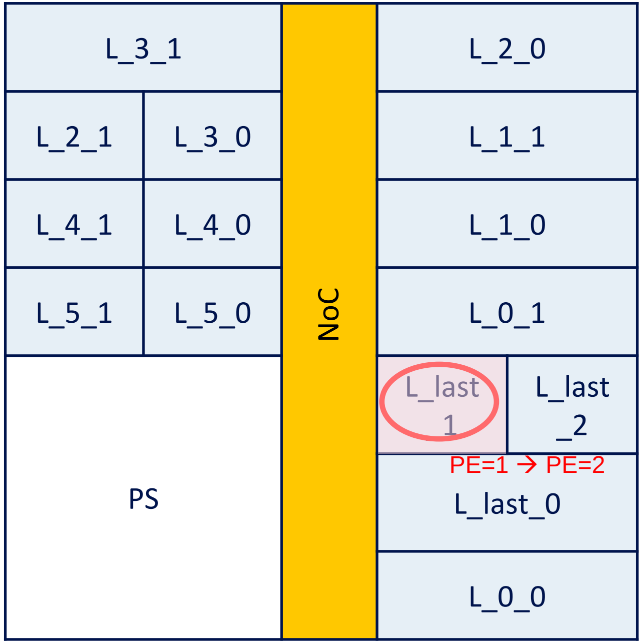
PE=1 → PE=2

<NoC-based system>

Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study

Example: CNN-2 benchmark



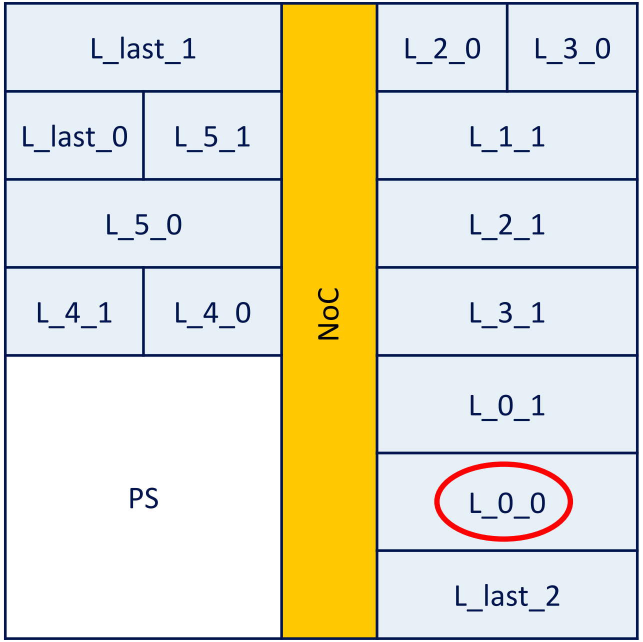
<NoC-based system>

And so on…

Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study

Example: CNN-2 benchmark



<NoC-based system>

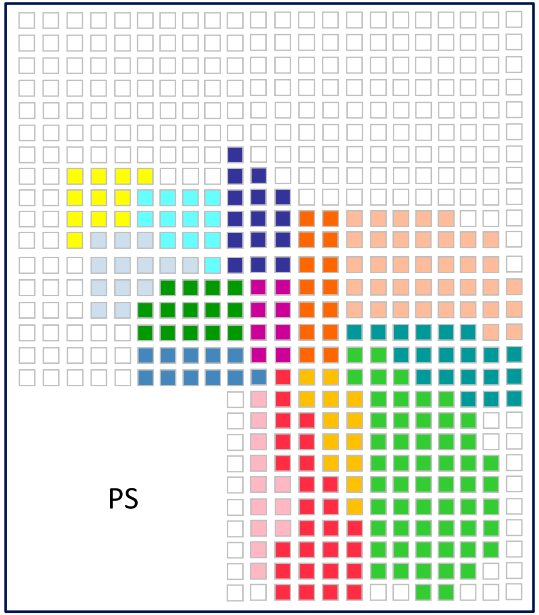
Already reached the final design point

➔ Migrate to monolithic flow

Idea – Incremental Refinement Strategy and Profiling

- Results: Design Space Exploration (DSE) case study

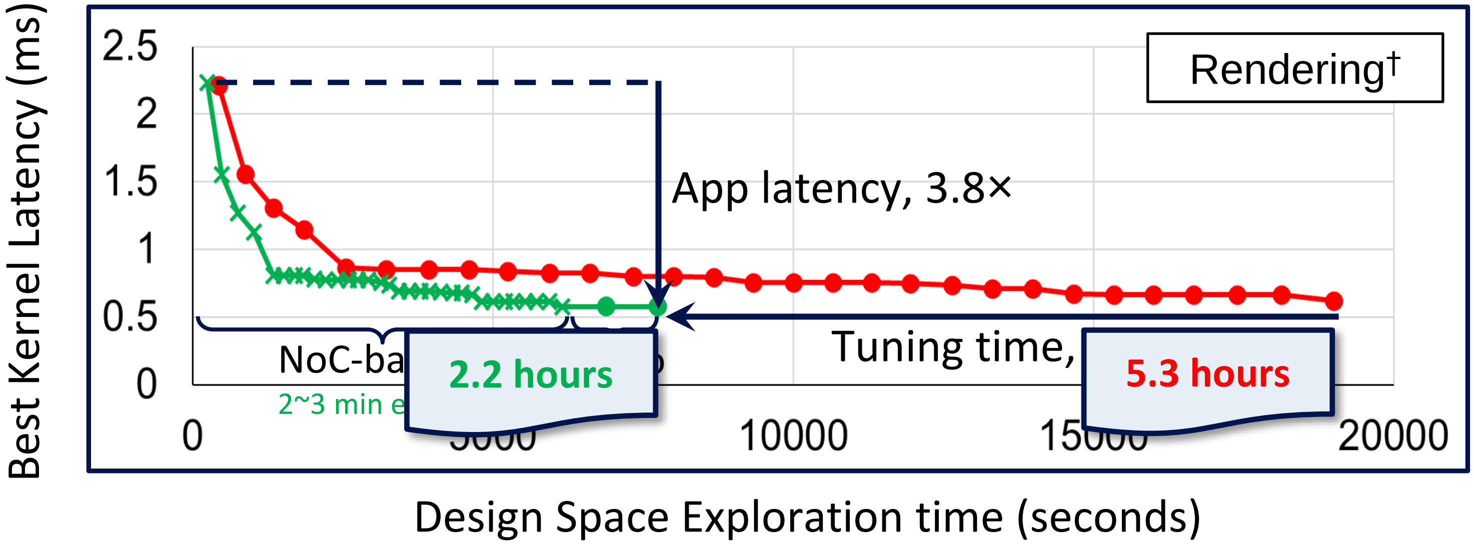
Example: CNN-2 benchmark



<Monolithic system – Just an illustration…>

- Wanted to show that 14 operators are monolithically compiled (slow)
- NoC is removed
- Continues to identify the bottleneck and refine until the design space is all explored

Idea – Incremental Refinement Strategy and Profiling



2.2 hours

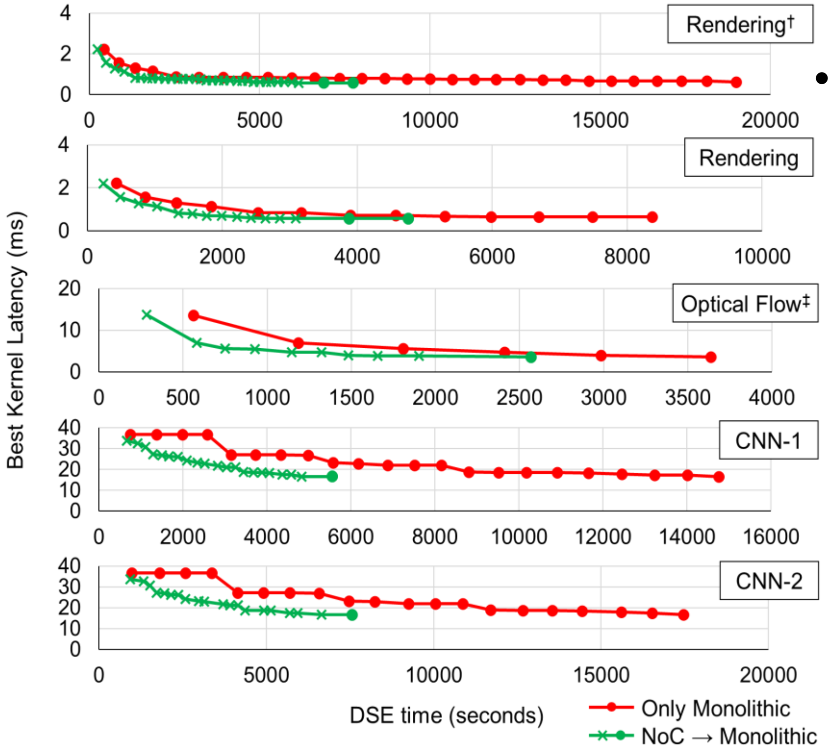
5.3 hours

● Monolithic system ✕ NoC-based system

🟥 Only Monolithic

🟩 NoC ⟶ Monolithic

Idea – Incremental Refinement Strategy and Profiling



<Selected DSE results: Our incr. refinement strategy vs Monolithic only>[5]

- Reduce tuning time by 1.3~2.7× while improving application latency by 2.2~12.7×

[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

Idea – Incremental Refinement Strategy and Profiling

- Advantages
  - Just like SW, we can quickly map the application on the FPGA, profile to find the bottleneck, and recompile only the functions that have changed
  - Faster tuning time is expected because initial design points are iterated with the fast separate compilation (2~3 min in some cases)
  - No loss in the performance for the final design

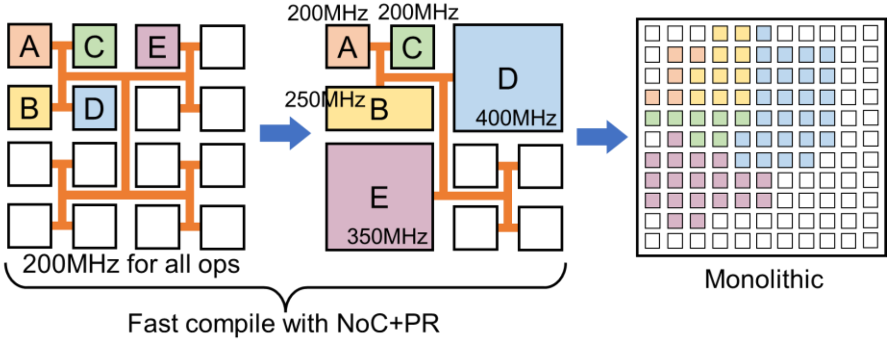


Table of Contents

- Motivation
- Idea – Separate compilation in Parallel using Partial Reconfiguration
- Idea – More Flexibility using Hierarchical PR
- Idea – Incremental Refinement Strategy and Profiling
- Discussion & Conclusion

Discussion & Conclusion

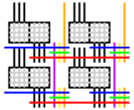
Slides removed for distribution

Discussion & Conclusion

- Soft NoC consumes FPGA resources
  - For all traffic patterns, is the current BFT NoC the best?
    - Some exploration for highly unbalanced traffic in [10]

- Conclusion
  - **SW-like Incremental Refinement FPGA development**
    - Fast Separate Compilation in Parallel using NoC + (Hierarchical) Partial Reconfiguration
    - Incremental Refinement strategy
    - Profiling using FIFO counters

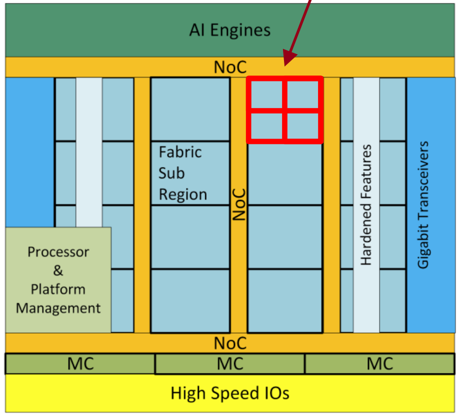
Thank you ☺



[10] Park et al., “Asymmetry in Butterfly Fat Tree FPGA NoC”, FPT 2023

Appendix

- Q. How is it related to FPGAs with hard NoC(e.g. AMD Versal)?
  - Can create similar hard NoC + PR pages platform
    - Limited NoC ports? Soft switch logic, Hierarchical PR pages
  - Can instantiate similar FIFO counter logic in NoC interfaces
  - Don’t need to migrate to monolithic system

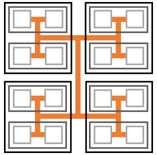


<Example Versal Floorplan[6]>

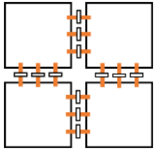
[6] I. Swarbrick et al., “Network-on-Chip Programmable Platform in Versal™ ACAP Architecture”, FPGA 2019

Appendix

- Q. How is it related to RapidWright from AMD Research?
  - RapidWright is an open source framework that enables netlist and implementation manipulation
  - Fast FPGA compilation work with RapidWright: [7,8,9]



  - PR is top-down, using a pre-routed overlay
    - Pro: don't need global stitching
    - Con: Requires NoC, NoC BW could be bottleneck
      - [11] doesn't use NoC but still uses PR. (switchbox PR pages)

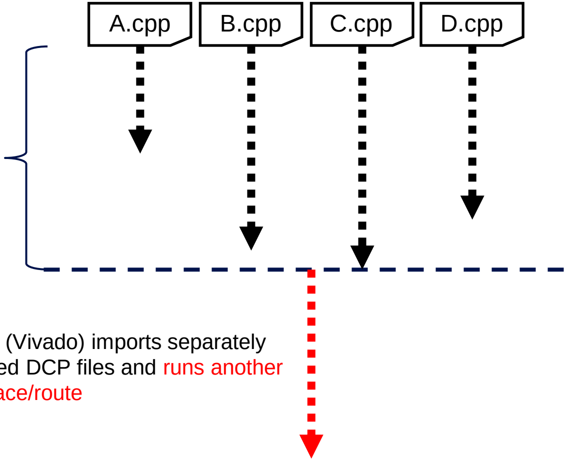


  - RapidWright, bottom-up, going through the global stitching
    - Pro: don't need NoC
    - Con: Requires global stitching
      - Fast routing challenge?

[7] Thomas et al., "Software-like Compilation for Data Center FPGA Accelerators", HEART 2021
[8] Guo et al., "RapidStream: Parallel Physical Implementation of FPGA HLS Designs", FPGA 2022
[9] Nguyen et al., "SPADES: A Productive Design Flow for Versal Programmable Logic", FPL 2023
[11] Xiao et al., "Fast linking of separately-compiled FPGA blocks without a NoC", FPT 2020

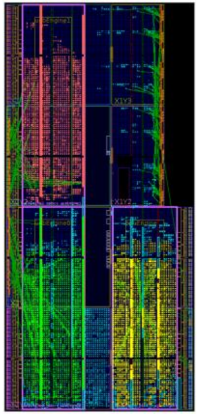
Appendix

- Q. Doesn’t Vivado support Out-of-Context flow? Without PR?
  - In synthesis, does save compile time.
    - HLS/Synthesize A.cpp, B.cpp, C.cpp, D.cpp
    - Then, stitch *.dcp ➔ Top-level stitching isn’t time-consuming
  - In implementation, does NOT save compile time.



Separate HLS/Synthesis/Place/Route

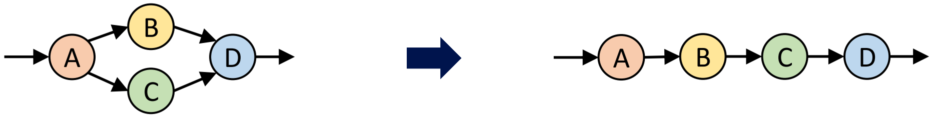
Vendor tool (Vivado) imports separately placed/routed DCP files and runs another top-level place/route

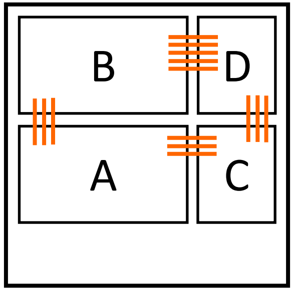


<Hierarchical Design Tutorial, ug946>

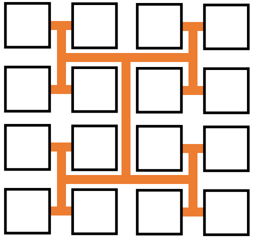
Appendix

- Q. Why do you need a NoC? Why not just PR pages?
  - Then, the static logic is application-specific
    ➔ Need a new static logic for each application?





?

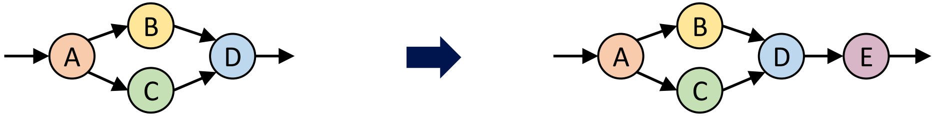


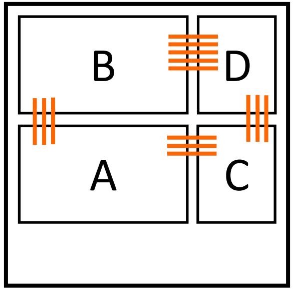
<NoC + PR pages>

<No NoC, only PR pages>

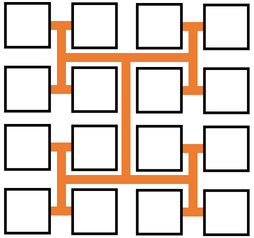
Appendix

- Q. Why do you need a NoC? Why not just PR pages?
  - Then, the static logic is application-specific
    ➔ Need a new static logic for each application?
    ➔ Can’t add new operator. Interconnection between operators can’t change





?

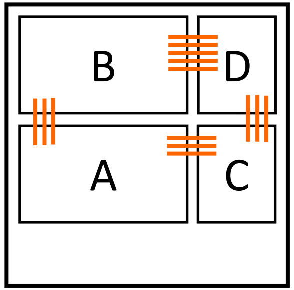


<NoC + PR pages>

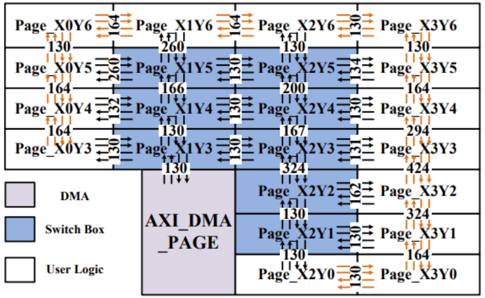
<No NoC, only PR pages>

Appendix

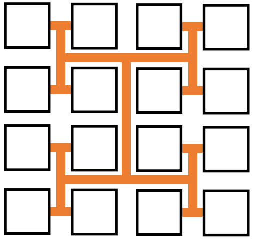
- Q. Why do you need a NoC? Why not just PR pages?
  - Then, the static logic is application-specific
    ➔ Need a new static logic for each application?
    ➔ Can’t add new operator. Interconnection between operators can’t change
  - If you are fixed with interconnections of operators, then possible![10]
  - Or with switchbox PR pages[11], possible! ➔ More wires



<No NoC, only PR pages>[10]



<SW PR pages + Logic PR pages>[11]



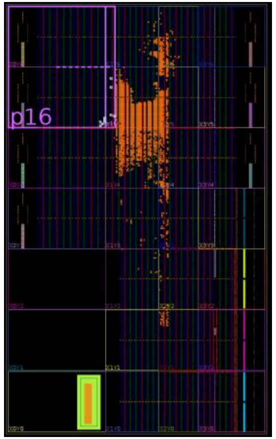
<NoC + PR pages>

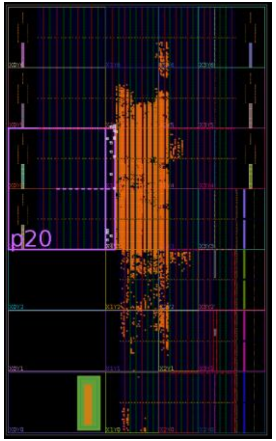
[10] Xiao et al., “HiPR: High-level partial reconfiguration for fast incremental FPGA compilation”, FPL 2022

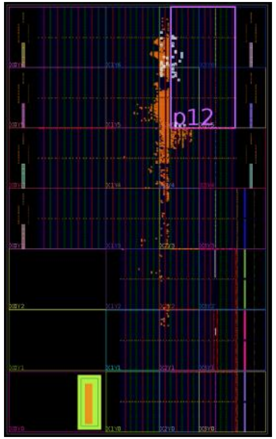
[11] Xiao et al., “Fast linking of separately-compiled FPGA blocks without a NoC”, FPT 2020

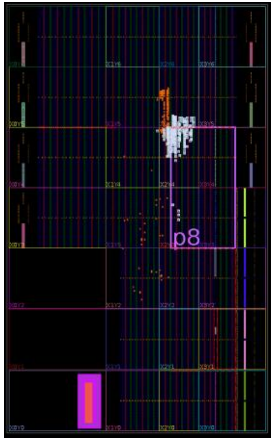
Appendix

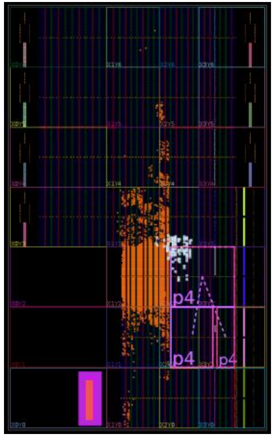
- Q. Some limitations on Vivado PR technology?
  - Abstract shell, not perfect
    - In [3], size of static design of abstract shell(quad page): 129 LUTs~15508 LUTs ➔ Had some workaround in [3]
    - Note: size of quad page is about 30K LUTs











[3] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

[5] Park et al., "REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs", FPGA 2024

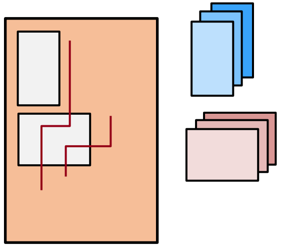
Appendix

- Q. Some limitations on Vivado PR technology?
  - Abstract shell, not perfect
    - In [3], size of static design of abstract shell(quad page): 129 LUTs~15508 LUTs ➔ Had some workaround in [3]
    - Note: size of quad page is about 30K LUTs
  - Static routing over reconfigurable regions
    - Addressed in [5]
  - Reconfigurable module relocation?
    - Note that in page assignment, if it needs to be moved to a different single-sized page, it needs to be newly placed/routed. ➔ Partial bitstreams can't be simply relocated

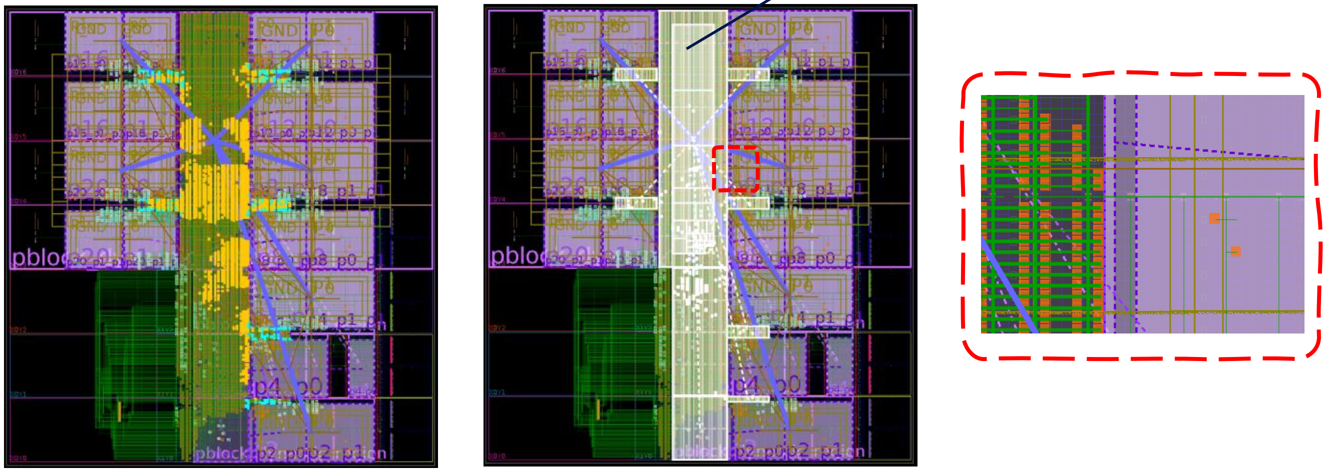
[3] Park et al., “Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration”, FPT 2022
[5] Park et al., “REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs”, FPGA 2024

Appendix

- Q. Some lir
  - Abstract
    - In [
      129
    - Not
  - Static ro
    - Add
  - Reconfig
    - Not
      diffe
      ➔ F



- In Vivado PR, static net can route over reconfigurable regions
- static ↔ reconfigurable: interface nets
- static ↔ static: can be prevented ➔ CONTAIN_ROUTING ON



Orange: NoC, Cyan: Pipeline regs

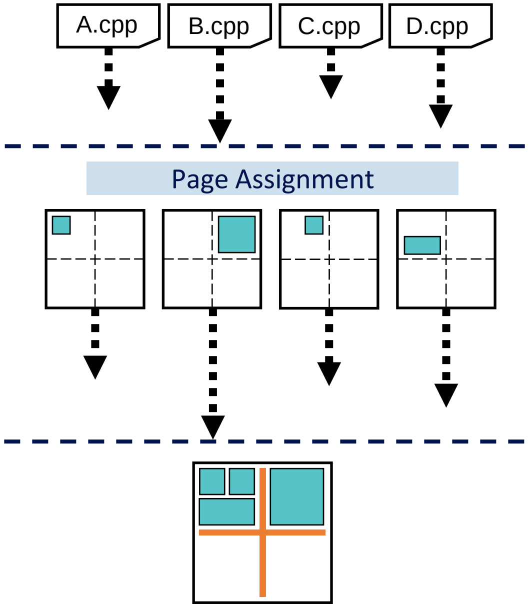
Static routing, PR

[3] Park et al., “Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration”, FPT 2022

[5] Park et al., “REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs”, FPGA 2024

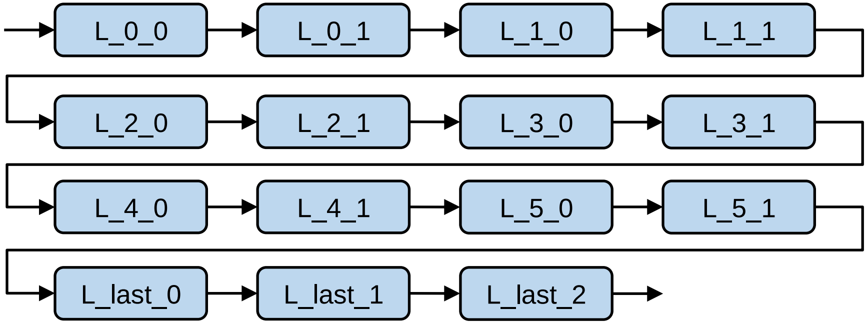
Appendix

- Q. How to determine whether a synthesized netlist fits in a PR page or not?
  - Irregular columnar resource distribution of FPGAs
  - AMD PR technology allows static routing to route over PR pages
  - Every design (netlist) has different routing complexity
    - E.g. 60% LUT util could fail in some designs while even 80% LUT util doesn’t fail in some designs
- Our solution
  - Per each PR page, **train a classifier that predicts whether a netlist can be successfully mapped or not**
  - Train input: a variety of designs with different resource util, Rent complexity, etc
  - Features: post-synthesis resource estimates, Rent value, average fanout, total instances



Appendix

- Q. How difficult is the designs decomposition?
  - For some designs, intuitive
  - For some designs, more challenging
  - Some of our benchmarks are from Rosetta HLS benchmark[3] that are not necessarily in dataflow form



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

Appendix

- Q. Final design point of our incremental strategy vs monolithic-only flow?
  - In our experiments, they reach to the similar final design points
  - But
    - sometimes the final design point of the NoC flow doesn't meet the timing in the monolithic flow
    - sometimes NoC flow fails earlier than the monolithic-only flow
    - sometime monolithic-only flow fails earlier than the NoC flow

    - Different implementation directives?