REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs

Dongjoon(DJ) Park, André DeHon

Implementation of Computation Group
University of Pennsylvania

Quickly test the current design on HW

Analyze the results

REFINE: Runtime Execution Feedback for INcremental Evolution on FPGA Designs

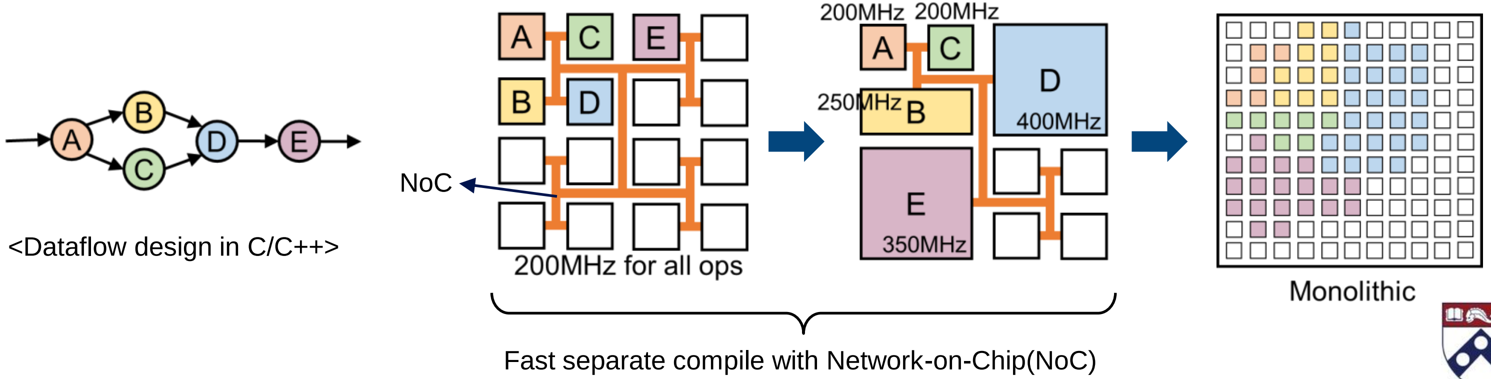
Select the next design point

Dongjoon(DJ) Park, André DeHon

Implementation of Computation Group
University of Pennsylvania

Story

- Problem: FPGA design optimization is challenging
  - Don't have good visibility into performance/bottlenecks of HW design
  - Design iterations are slow due to long FPGA compile times
- Goal: Make FPGA more like SW
  - Profiling: Bottleneck identification based on FIFO full/empty counters
  - Fast Separate Incremental Refinement strategy for FPGA designs



Story

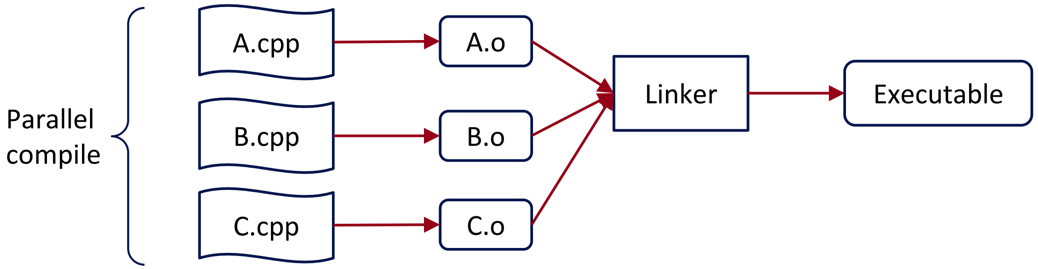
- Problem: FPGA design optimization is challenging
  - Don't have good visibility into performance/bottlenecks of HW design
  - Design iterations are slow due to long FPGA compile times
- Goal: Make FPGA more like SW
  - Profiling: Bottleneck identification based on FIFO full/empty counters
  - Fast Separate Incremental Refinement strategy for FPGA designs
- What our framework will deliver
  - Identifies the bottleneck of the application along with the fast separate compilation
    ➔ SW-like FPGA development
- Result
  - Reduces design tuning time by 1.3~2.7× compared to Vivado's monolithic flow while achieving same performance

Table of Contents

Table of Contents

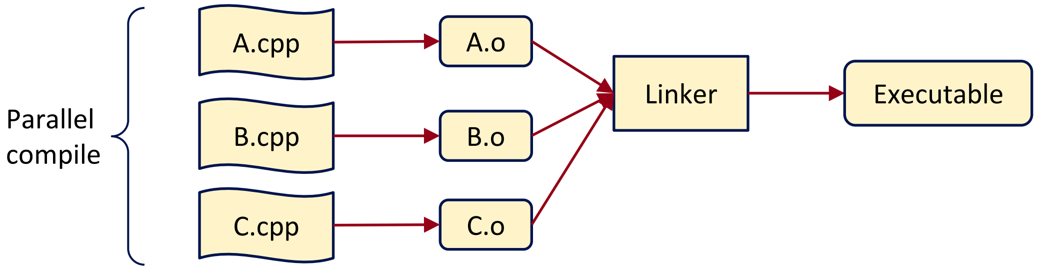
Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement



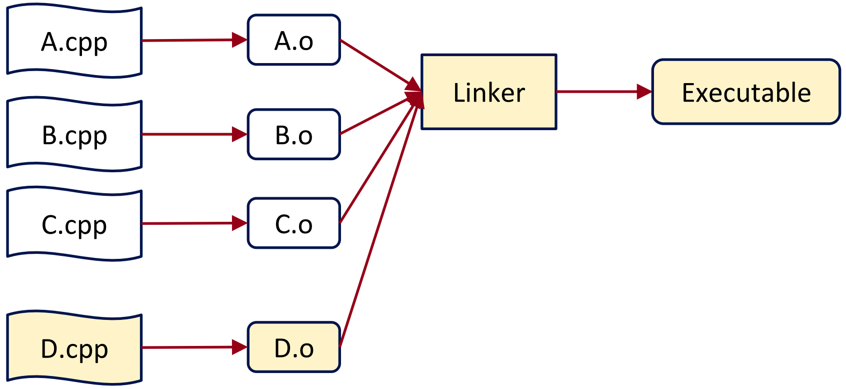
Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement



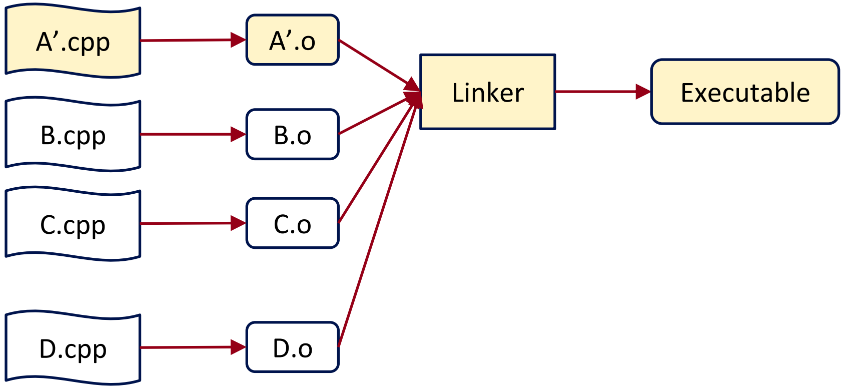
Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement



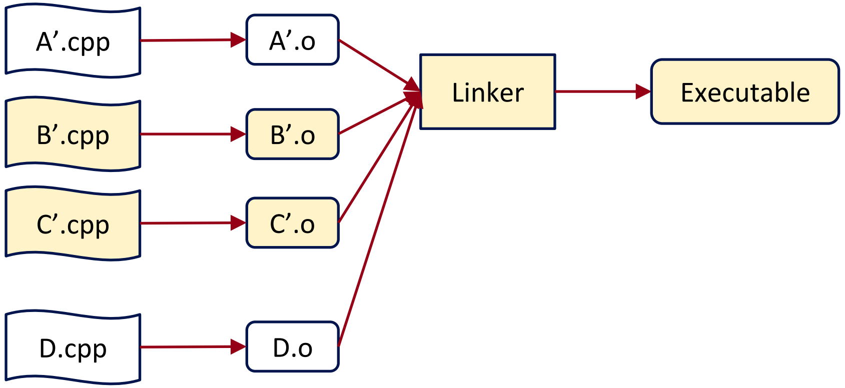
Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement



Motivation

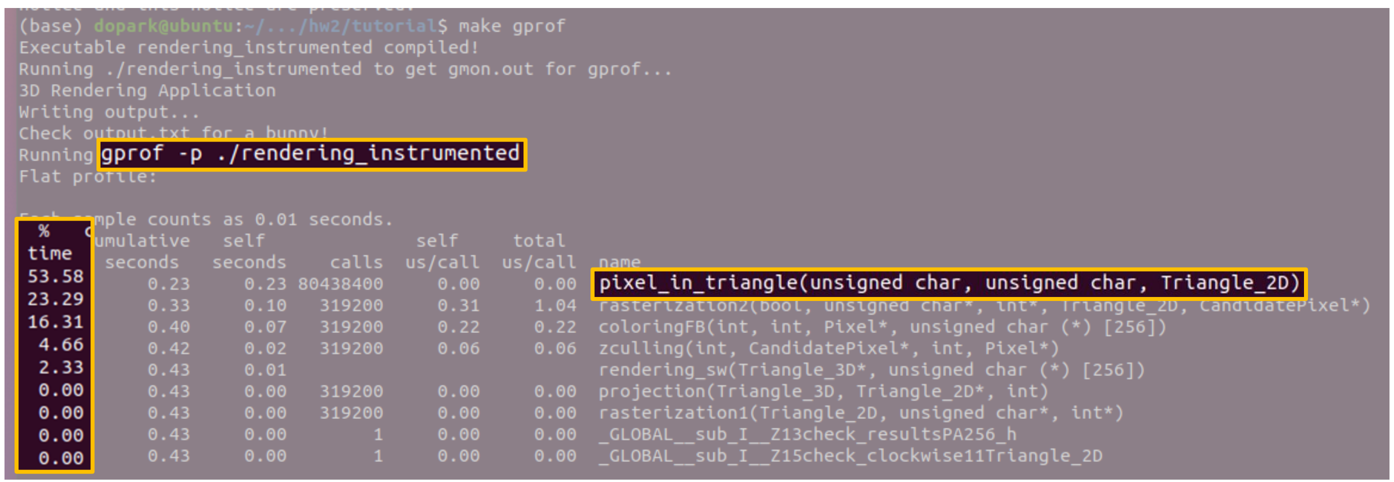
- How's SW development?
  1) Parallel compile, Incremental Refinement



Motivation

- How's 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.



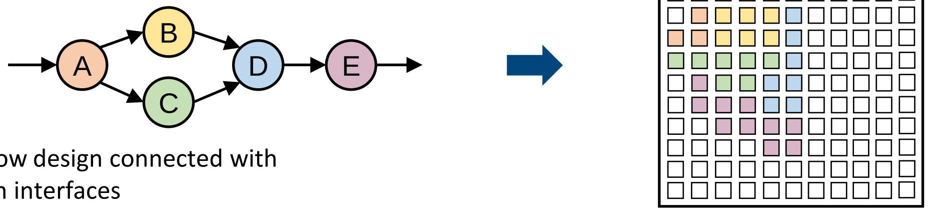
```
notice and this notice are preserved.
(base) dopark@ubuntu:~/.../hw2/tutorial$ make gprof
Executable rendering_instrumented compiled!
Running ./rendering_instrumented to get gmon.out for gprof...
3D Rendering Application
Writing output...
Check output.txt for a bunny!
Running gprof -p ./rendering_instrumented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

Penn Engineering
University of Pennsylvania

Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?



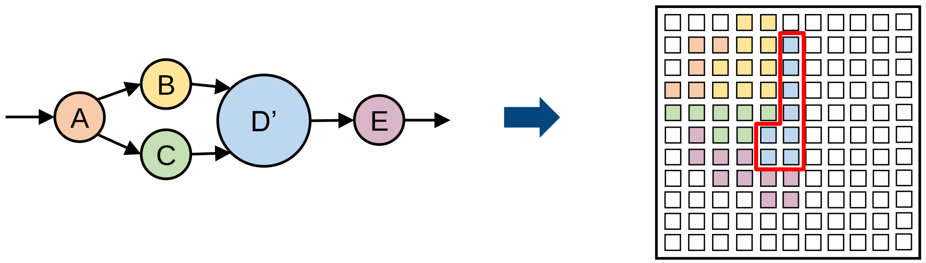
HLS dataflow design connected with
HLS stream interfaces

Q. Can we compile each function in parallel?

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

Motivation

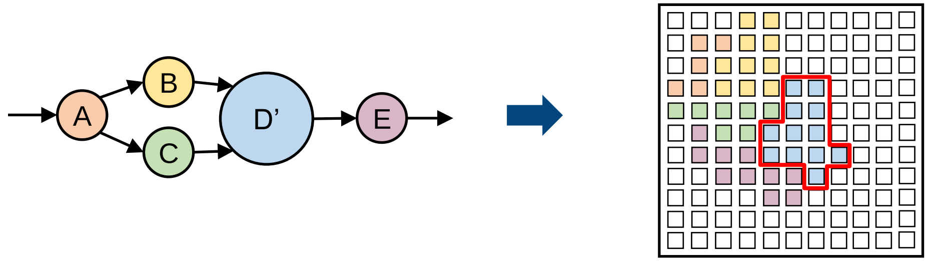
- How's SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?



Q. Can we recompile only the changed part?

Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?

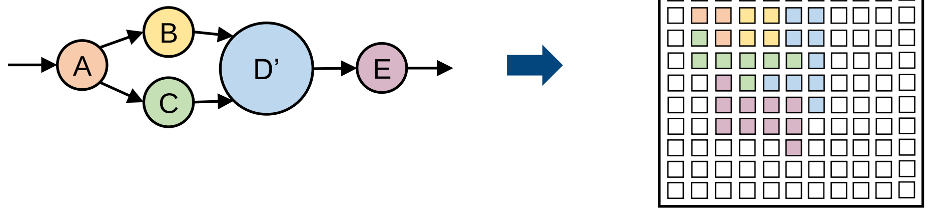


Q. Can we recompile only the changed part?

Something like this!

Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?

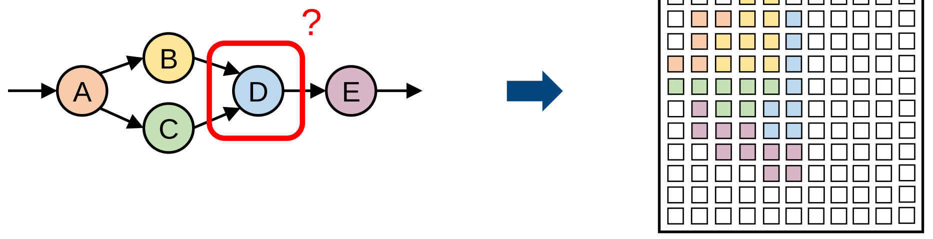


Q. Can we recompile only the changed part?

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

Motivation

- How's SW development?
  1) Parallel compile, Incremental Refinement
  2) Rich profiling tools
- How's current HW development?
  1) Parallel compile? Incremental Refinement?
  2) Profiling?



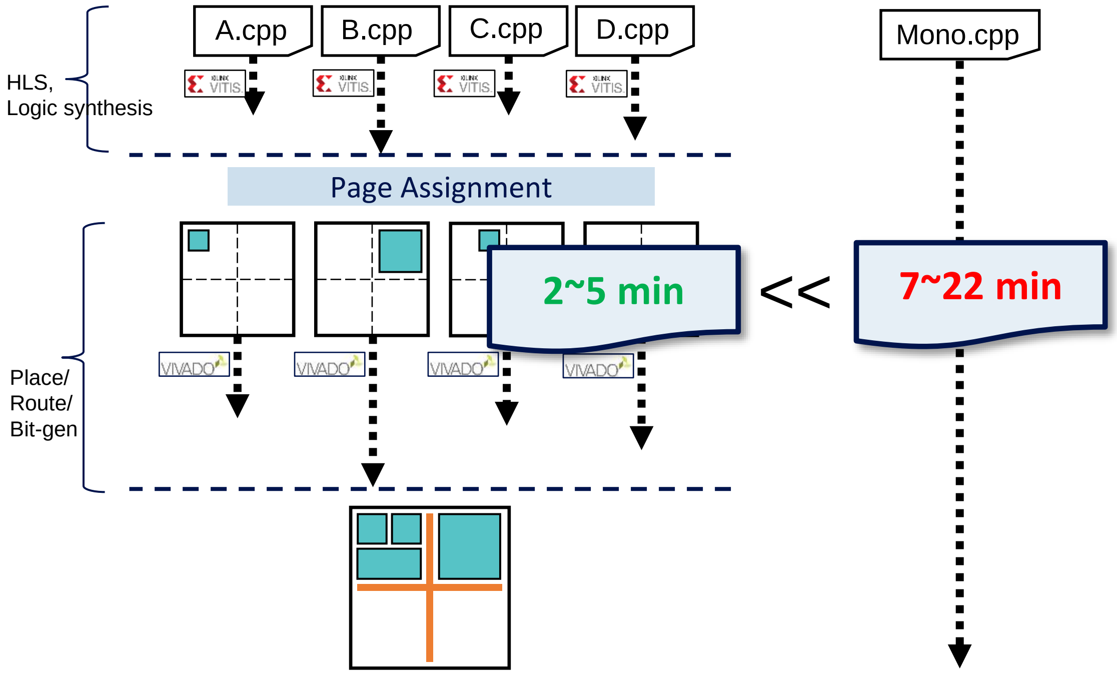
Q. How do we know which module to refine next?

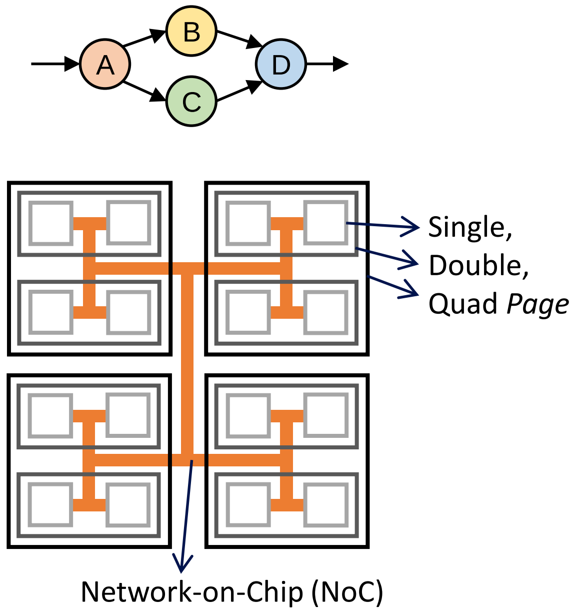
A. It's difficult to identify the bottleneck
➔ Lack of visibility on the inner state of the HW design

Table of Contents

Background

- Fast separate compilation using NoC and Partial Reconfig.(PR)[1]
  - Parallel compile, incremental compile for FPGA designs

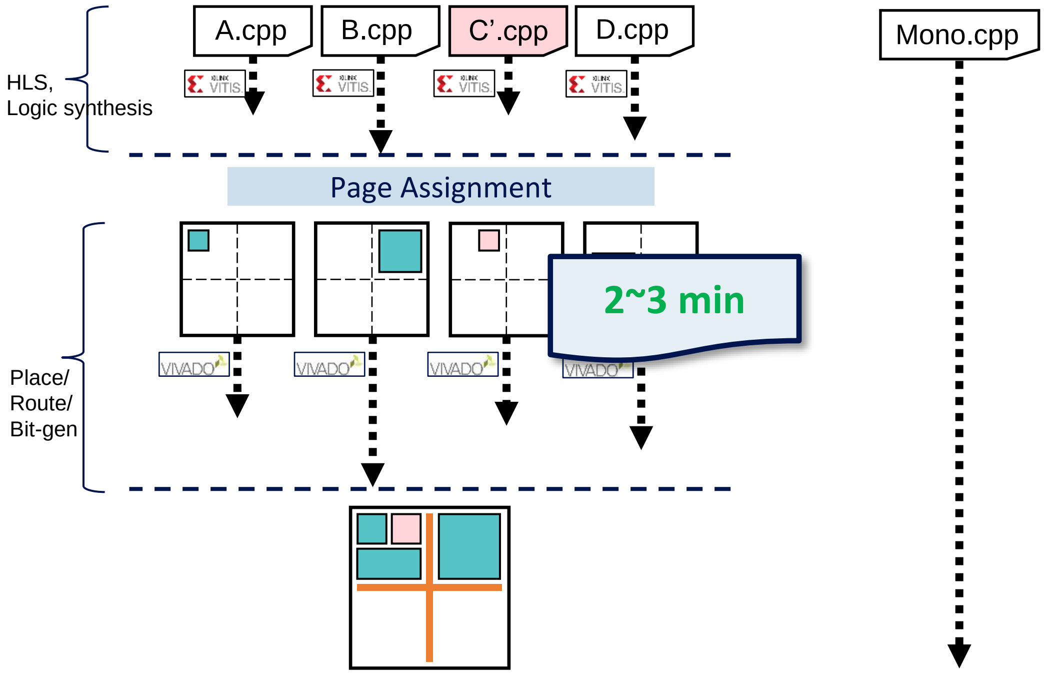


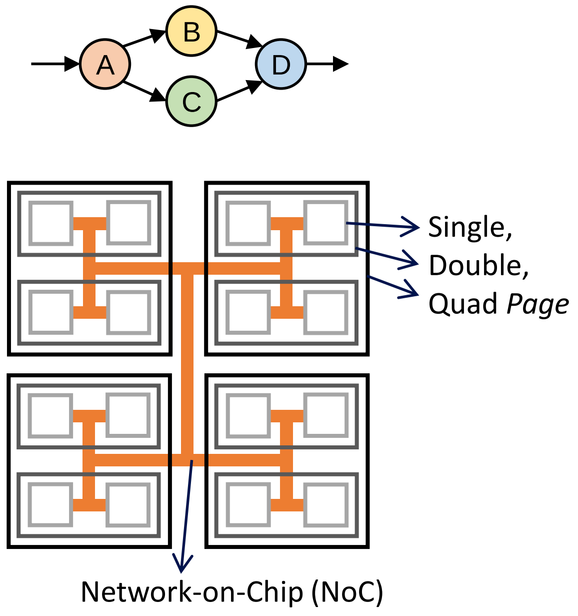


[1] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Background

- Fast separate compilation using NoC and Partial Reconfig.(PR)[1]
  - Parallel compile, incremental compile for FPGA designs





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

Table of Contents

Idea

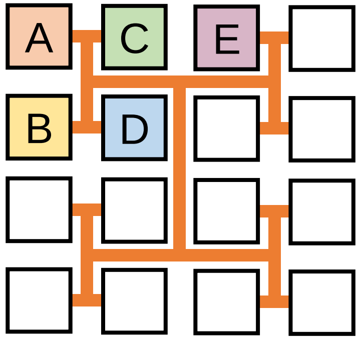
1) Fast separate incremental refinement strategy on FPGA designs
2) Profiling
   ➔ Bottleneck identification using FIFO counters
3) Incremental refinement
   ➔ Separate compilations in parallel using NoC and Partial Reconfiguration (PR)
   ➔ Enhancements to the previously proposed framework

Idea

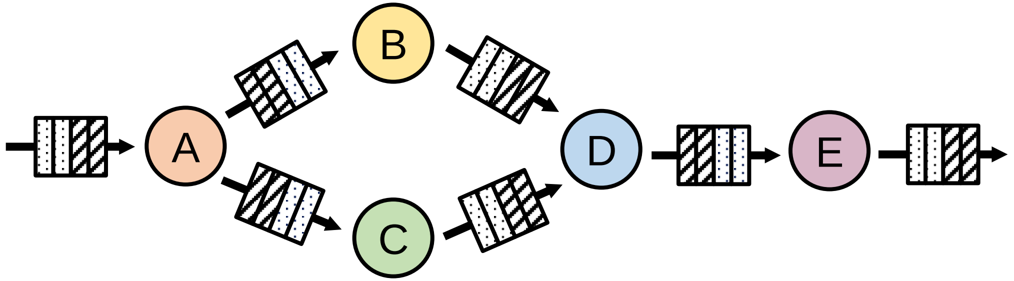
1) Fast separate incremental refinement strategy on FPGA designs
2) Profiling
   - ➔ Bottleneck identification using FIFO counters
3) Incremental refinement
   - ➔ Separate compilations in parallel using NoC and Partial Reconfiguration (PR)
   - ➔ Enhancements to the previously proposed framework

Idea – Incremental refinement strategy

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

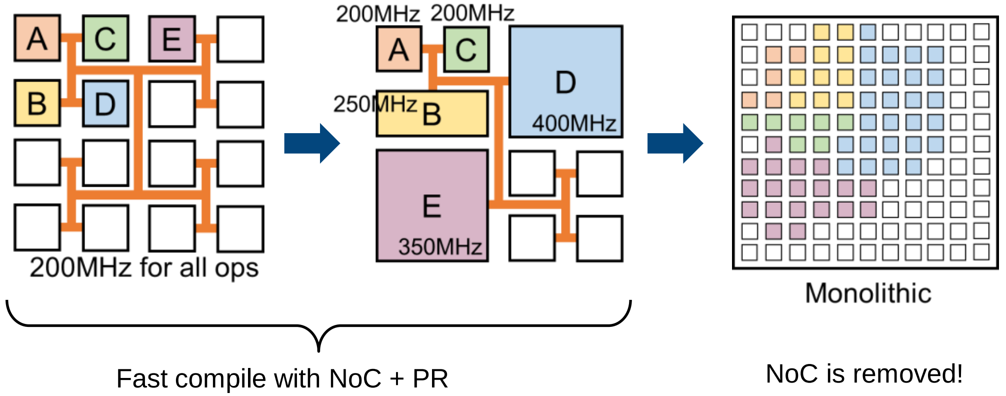


- Monolithic system
  - Pro: No NoC overhead
  - Con: Slow compile



Idea – Incremental refinement strategy

- Idea: Fast incremental refinement strategy
  - 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



Idea

1) Fast separate incremental refinement strategy on FPGA designs
2) Profiling
   ➔ Bottleneck identification using FIFO counters
3) Incremental refinement
   ➔ Separate compilations in parallel using NoC and Partial Reconfiguration (PR)
   ➔ Enhancements to the previously proposed framework

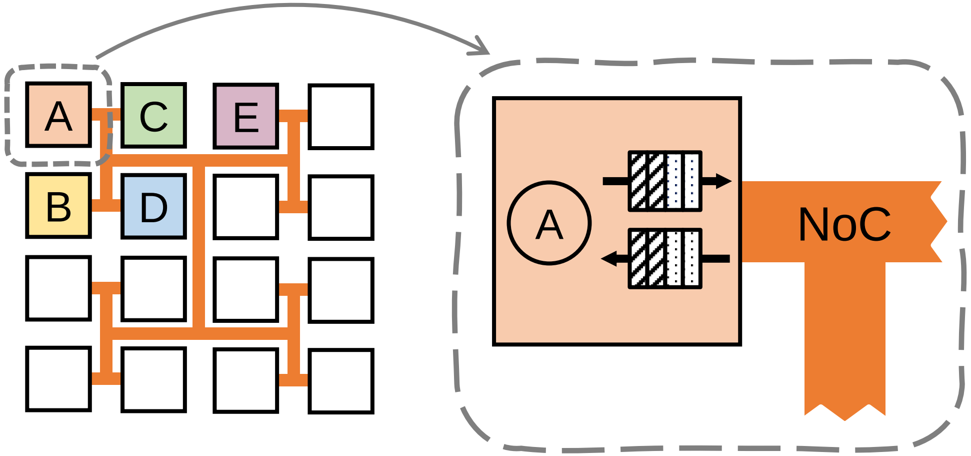
Penn Engineering
University *of* Pennsylvania

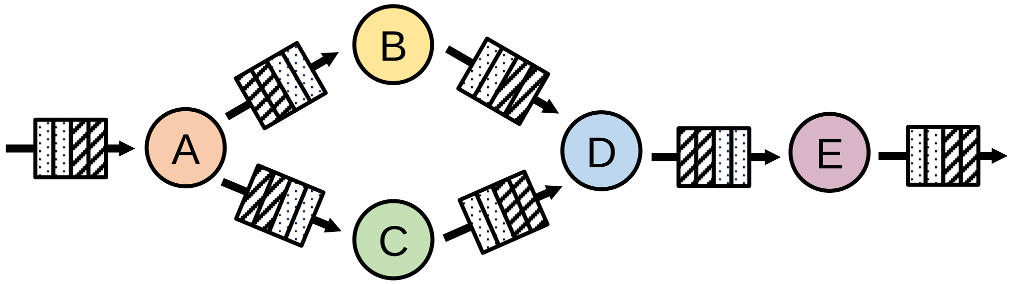
Idea – Profiling

Recall!

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

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



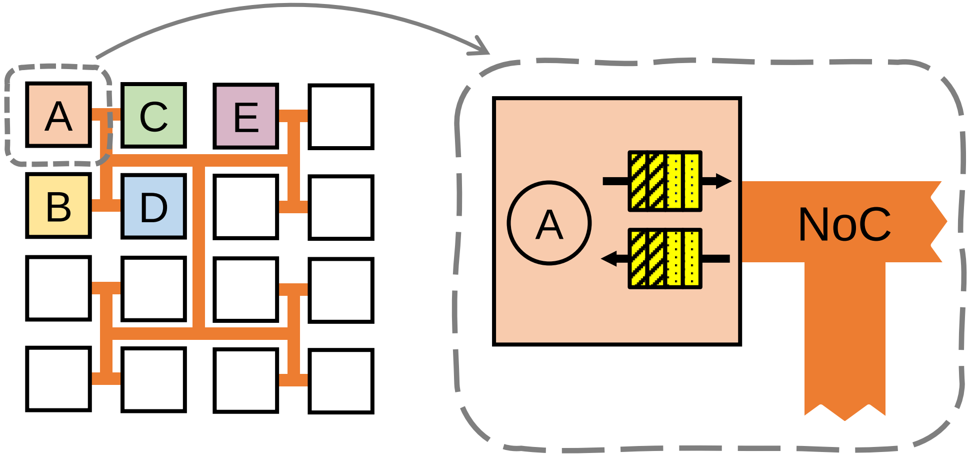


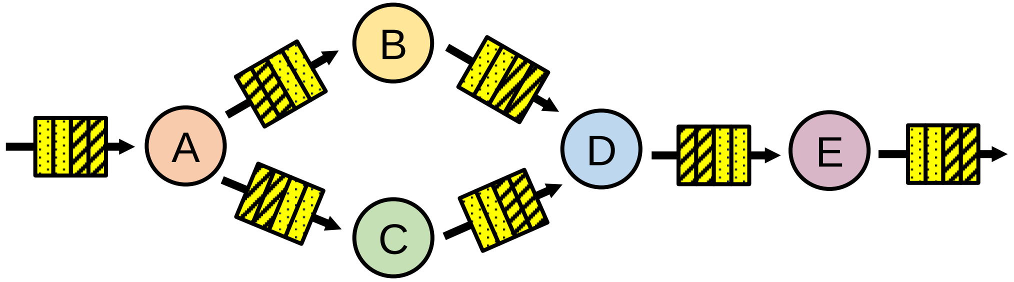
Idea – Profiling

Recall!

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

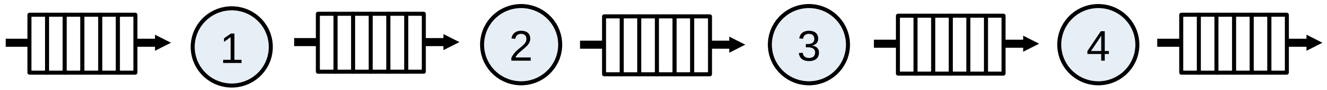
- Monolithic system
  - Pro: No NoC overhead
  - Con: Slow compile





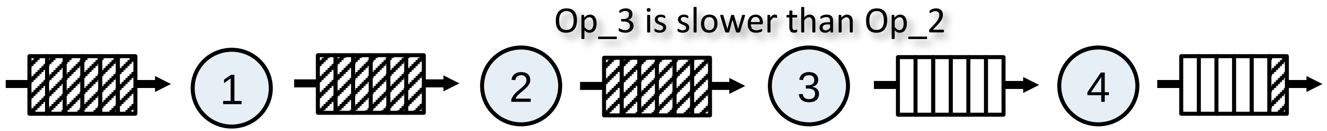
Idea – Profiling

- High-level intuition



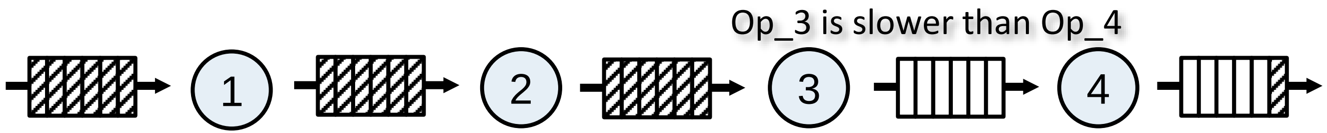
Idea – Profiling

- High-level intuition



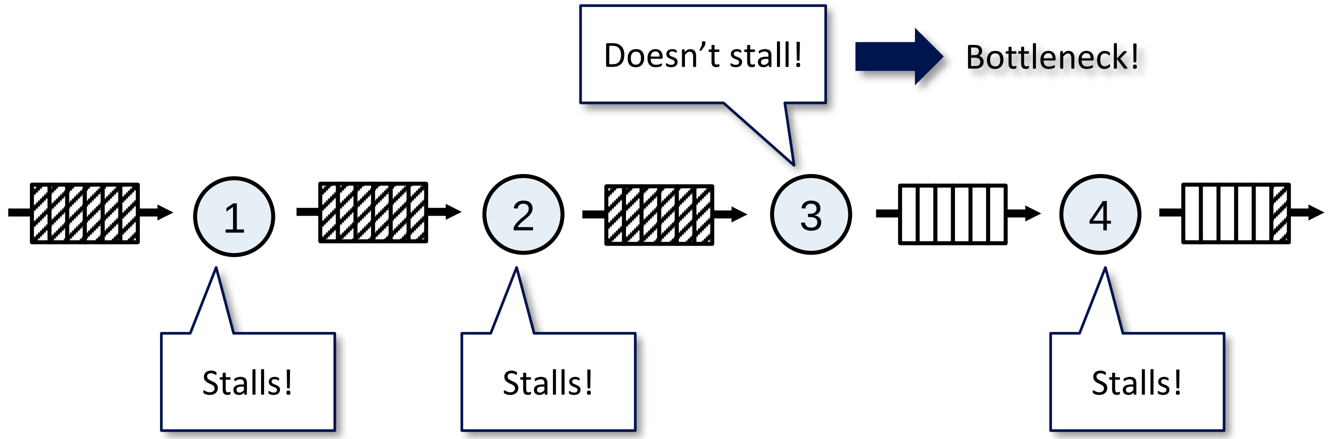
Idea – Profiling

- High-level intuition



Idea – Profiling

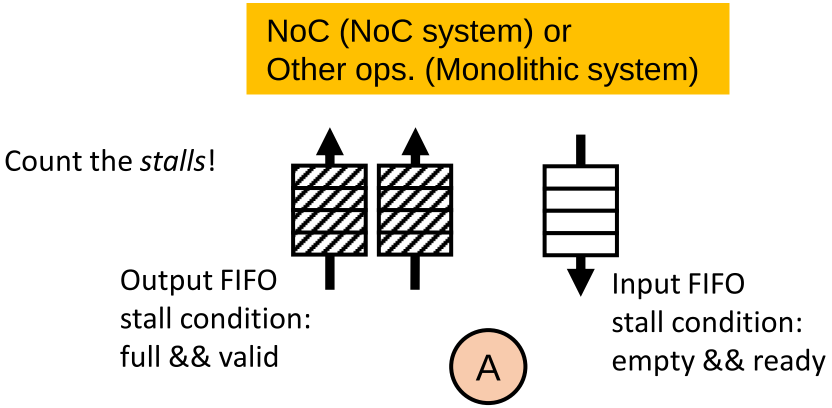
- High-level intuition



Idea – Profiling

- Use FIFO counters to identify

1) bottleneck operator
→ embedded in both NoC system, monolithic system

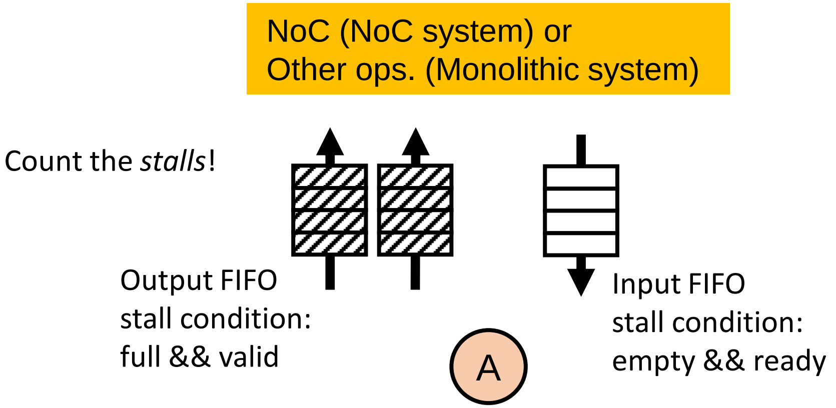


➔ **Op with the least stall counts may be the bottleneck**

Idea – Profiling

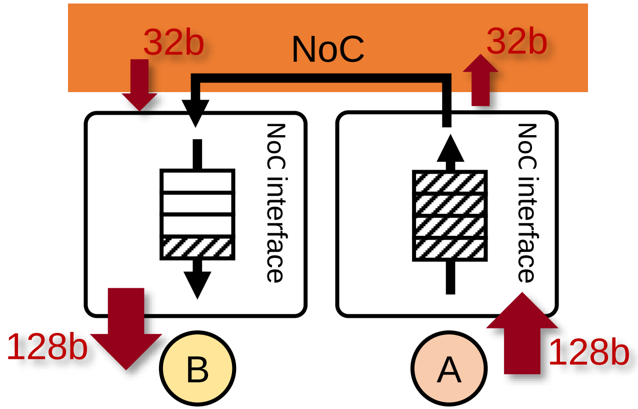
- Use FIFO counters to identify

1) bottleneck operator
→ embedded in both NoC system, monolithic system



➔ **Op with the least stall counts may be the bottleneck**

2) NoC bandwidth bottleneck
→ embedded in only NoC system

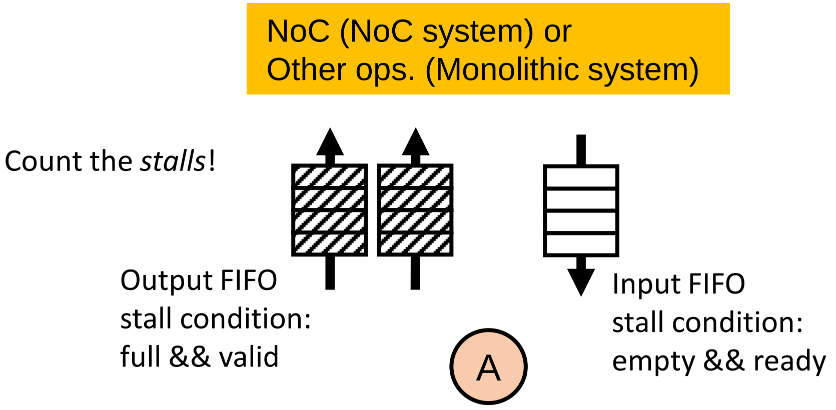


- Harms application performance
- Wrong bottleneck operator can be identified

Idea – Profiling

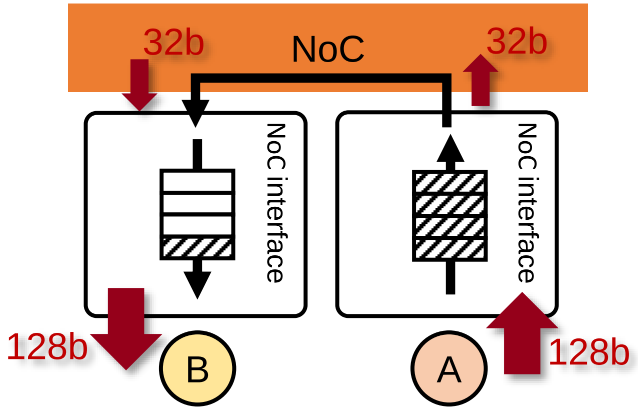
- Use FIFO counters to identify

1) bottleneck operator
→ embedded in both NoC system, monolithic system



➔ **Op with the least stall counts may be the bottleneck**

2) NoC bandwidth bottleneck
→ embedded in only NoC system



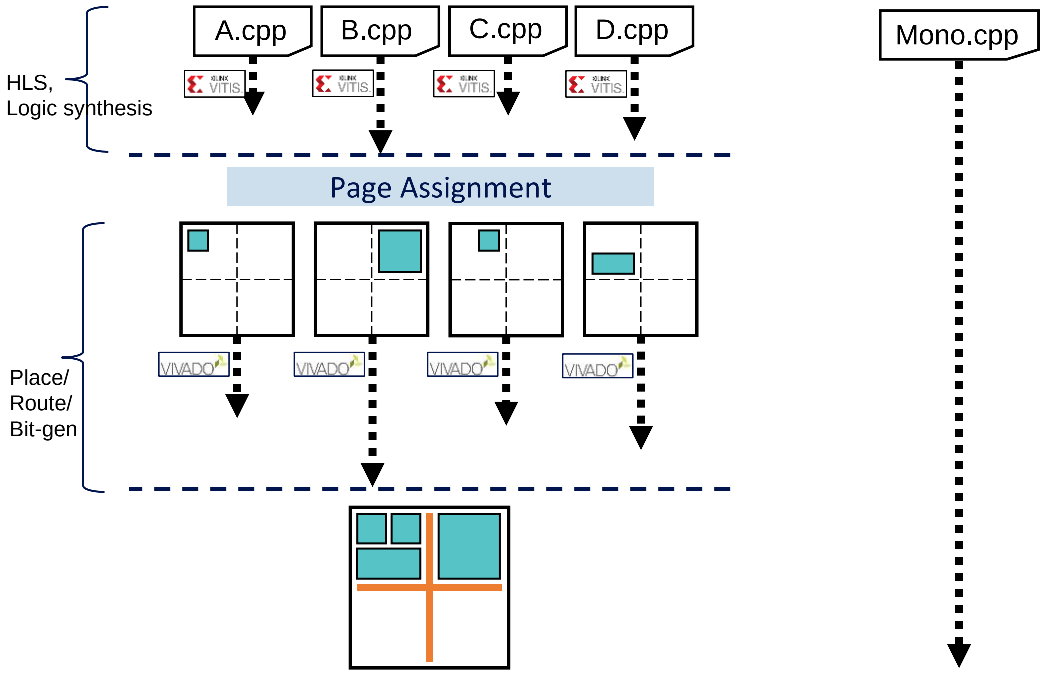
If A's Output FIFO's full↑ && B's Input FIFO's full↓
➔ **NoC bandwidth may be the bottleneck**

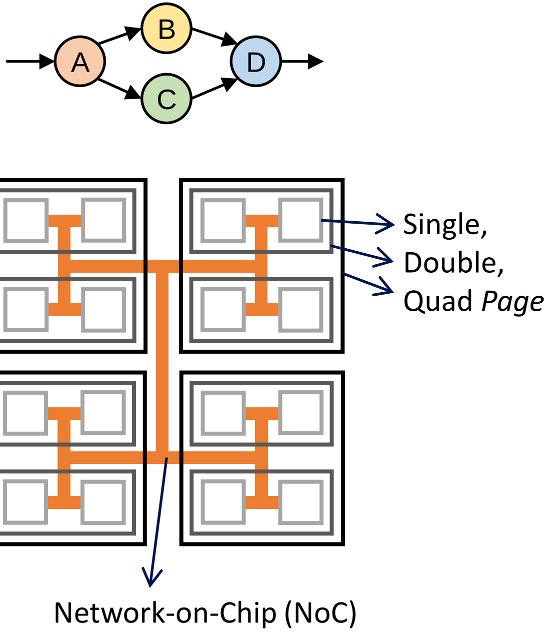
Idea

1) Fast separate incremental refinement strategy on FPGA designs
2) Profiling
   ➔ Bottleneck identification using FIFO counters
3) Incremental refinement
   ➔ Separate compilations in parallel using NoC and Partial Reconfiguration (PR)
   ➔ Enhancements to the previously proposed framework

Idea – Enhancements in NoC-based system

- Use Fast&Flexible/FPT'22[1]'s separate compilation framework



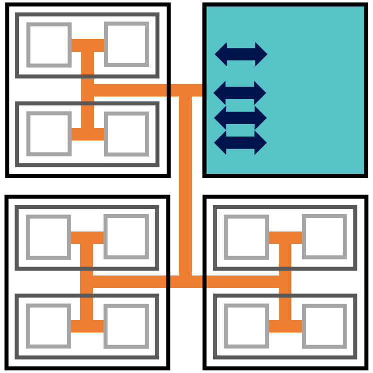


[1] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Idea – Enhancements in NoC-based system

- Use Fast&Flexible/FPT'22[1]'s separate compilation framework

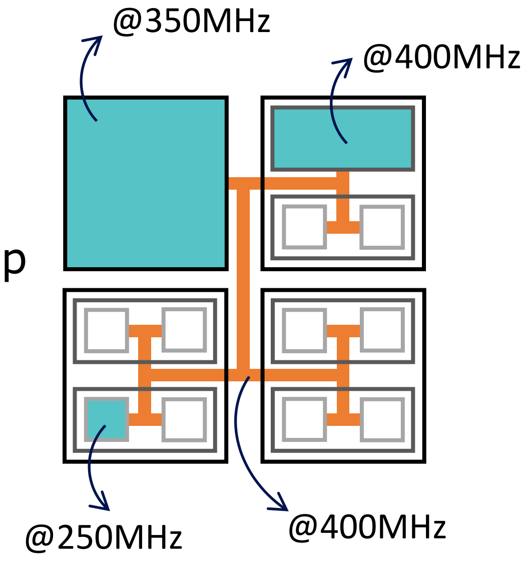
- This work enhances [1]'s NoC-based system
  - Mitigate NoC bandwidth bottleneck
    - Use multiple NoC interfaces



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

Idea – Enhancements in NoC-based system

- Use Fast&Flexible/FPT'22[1]'s separate compilation framework
- This work enhances [1]'s NoC-based system
  - Mitigate NoC bandwidth bottleneck
    - Use multiple NoC interfaces
  - Support for multiple clock frequencies for each op
    - NoC runs @ 400MHz,
    - Operators run @ 200~400MHz



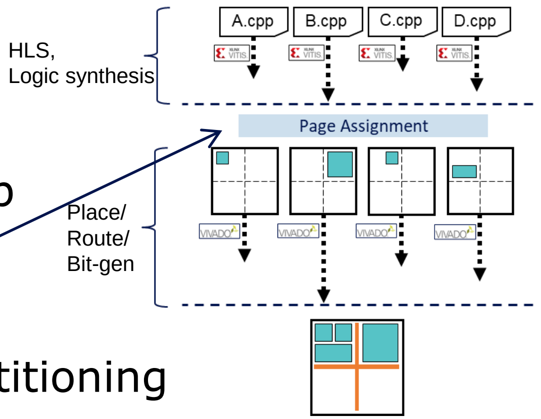
[1] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Idea – Enhancements in NoC-based system

- Use Fast&Flexible/FPT'22[1]'s separate compilation framework

- This work enhances [1]'s NoC-based system
  - 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

  - Other enhancements detailed in the paper

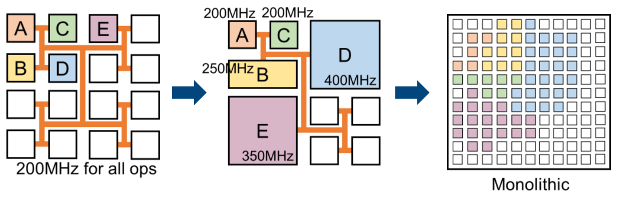


[1] Park et al., "Fast and Flexible FPGA Development using Hierarchical Partial Reconfiguration", FPT 2022

Table of Contents

Evaluation

- Profiling and fast separate compilation framework allow the developer/tools to quickly iterate:
  - Refine the design
  - Get updated performance and bottleneck



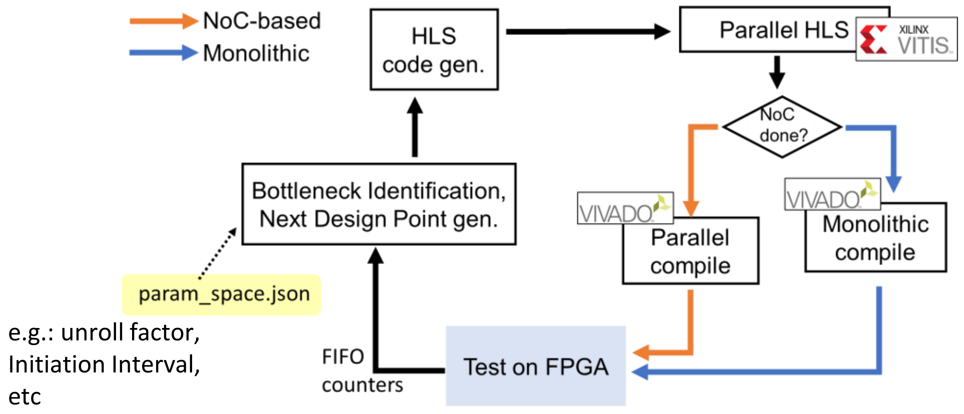
<Fast Incremental Refinement strategy>

- 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

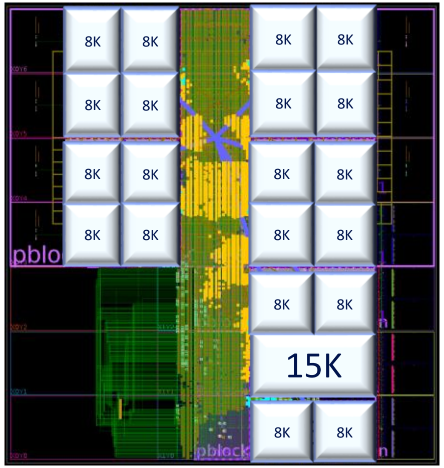
Evaluation

- 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



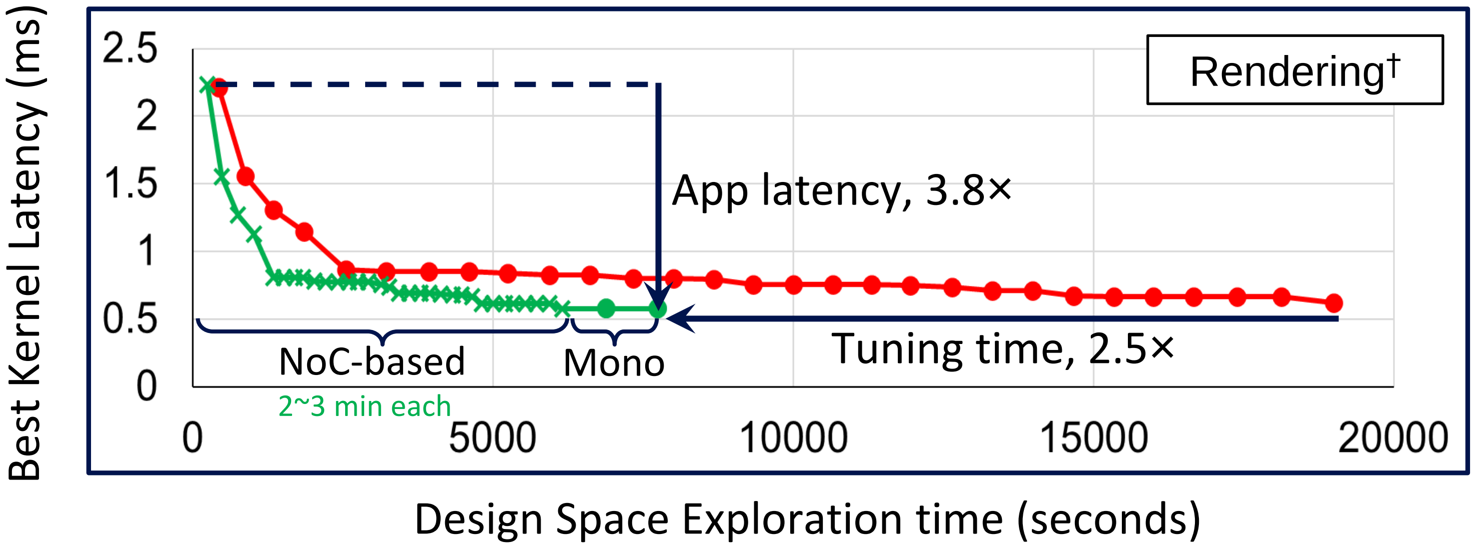
<Automated DSE experiment overview>



<NoC-based system overlay>

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

Evaluation

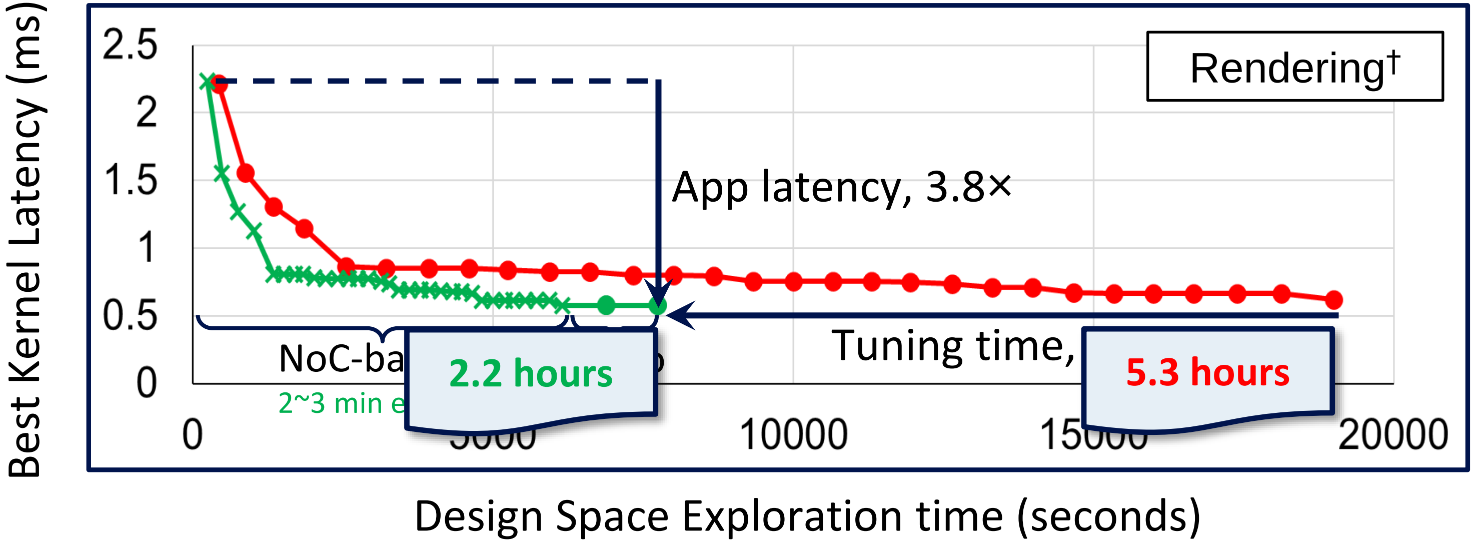


● Monolithic system ✕ NoC-based system

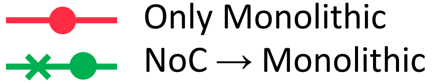
Only Monolithic

NoC ⟶ Monolithic

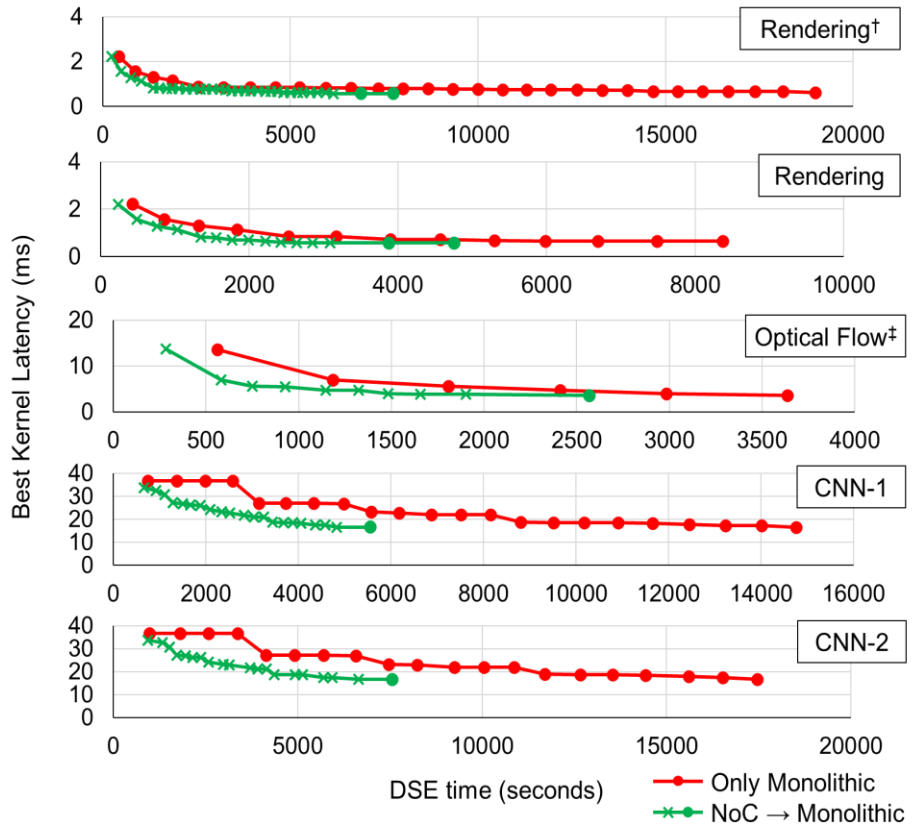
Evaluation



● Monolithic system ✕ NoC-based system



Evaluation



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

- Reduce tuning time by 1.3~2.7× while improving application latency by 2.2~12.7×
- Full results in the paper

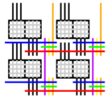
Table of Contents

Penn Engineering
UNIVERSITY of PENNSYLVANIA

Conclusion

- **SW development**
  - Rich profiling tools
  - Incremental refinement: secs, mins of compile cycles
- **SW-like FPGA development**
  - Bottleneck identification using FIFO counters to profile the HW design
  - Fast incremental refinement strategy
    - Starts with the NoC-based system: 2~3 min of compile cycles
    - Migrates to the monolithic system: longer compile cycles
    - Results: DSE case studies
      - Monolithic-only: 2~5 hrs
      - Our strategy: 1~2 hrs (1.3-2.7x faster), achieving same quality final monolithic design

Thank you ☺



Penn Engineering
University of Pennsylvania