Suppressing Spurious Dynamism of Dataflow Circuits via Latency and Occupancy Balancing

Jiahui Xu
ETH Zurich
Zurich, Switzerland

Lana Josipović
ETH Zurich
Zurich, Switzerland

Suppressing Spurious Dynamism of Dataflow Circuits

HS25 Semester Project by Ziad Malik

Background and Motivation

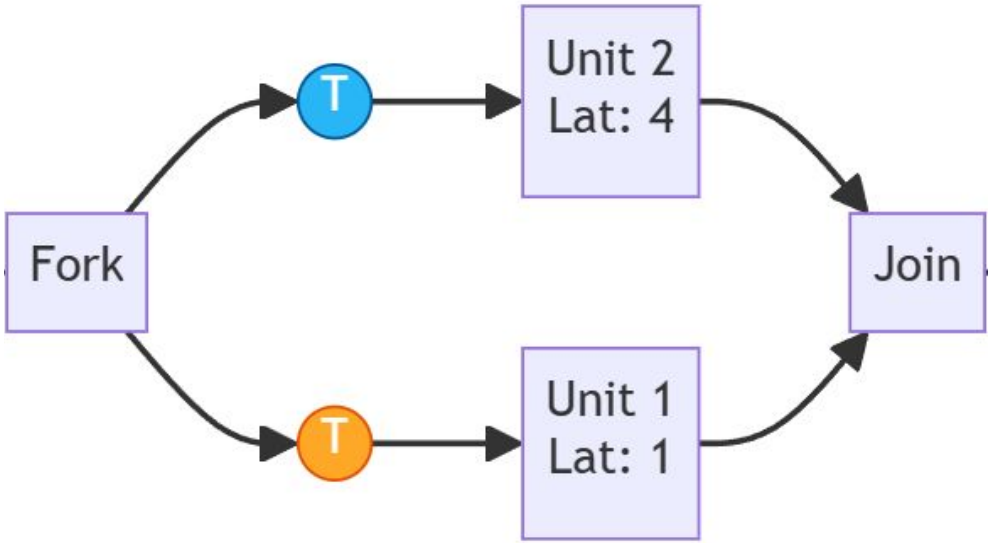
- Dynamatics' handshake mechanism provides incredible flexibility…
  - It helps us adapt execution at runtime.
  - Lets us deal with unpredictable control-flow graphs and irregular memory-accesses.

- … but it is also incredibly resource-intensive.
  - It can take up over 50% of resource usage.
  - It is widely employed within the circuit, regardless of whether a given circuit sub-graph can benefit from the flexibility of runtime execution it offers.

- Optimization is in order!
  - Idea: wherever data is predictably exchanged between units, remove handshakes!

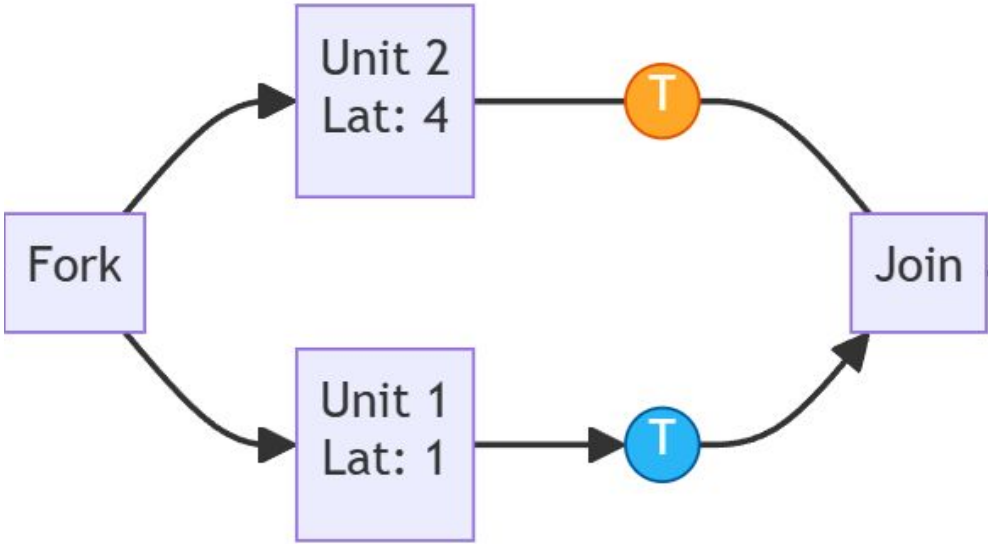
Major Optimization Hurdle:

- When data is forked for parallel processing, the dynamic nature of the circuit allows for data items to arrive at arbitrary times at their synchronization point:



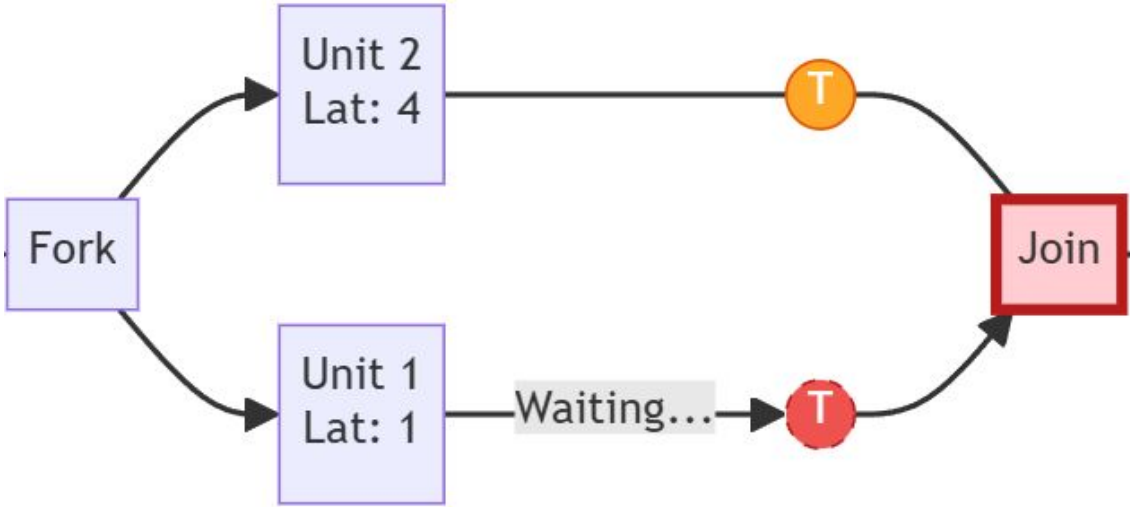
Major Optimization Hurdle:

- When data is forked for parallel processing, the dynamic nature of the circuit allows for data items to arrive at arbitrary times at their synchronization point:



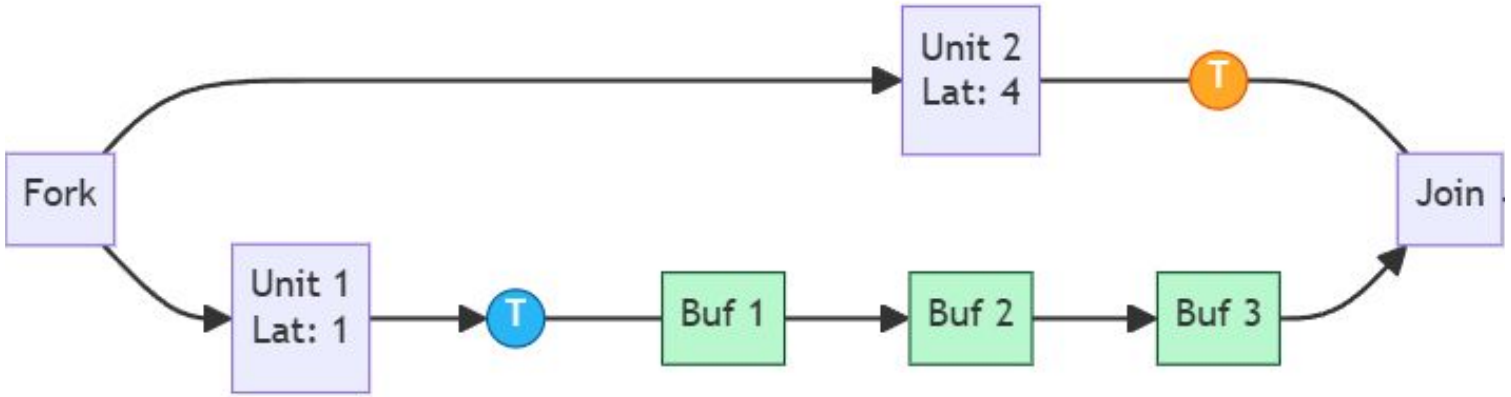
Major Optimization Hurdle:

- When data is forked for parallel processing, the dynamic nature of the circuit allows for data items to arrive at arbitrary times at their synchronization point:



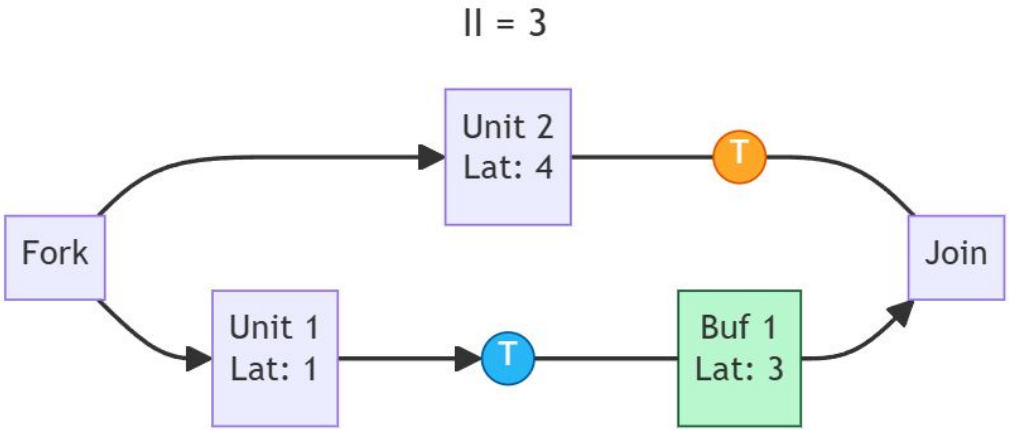
Naive Solution: Just Add Latency!

- Just identify the latency imbalance in a reconvergent path and place buffers on the fast path until we eliminate it.



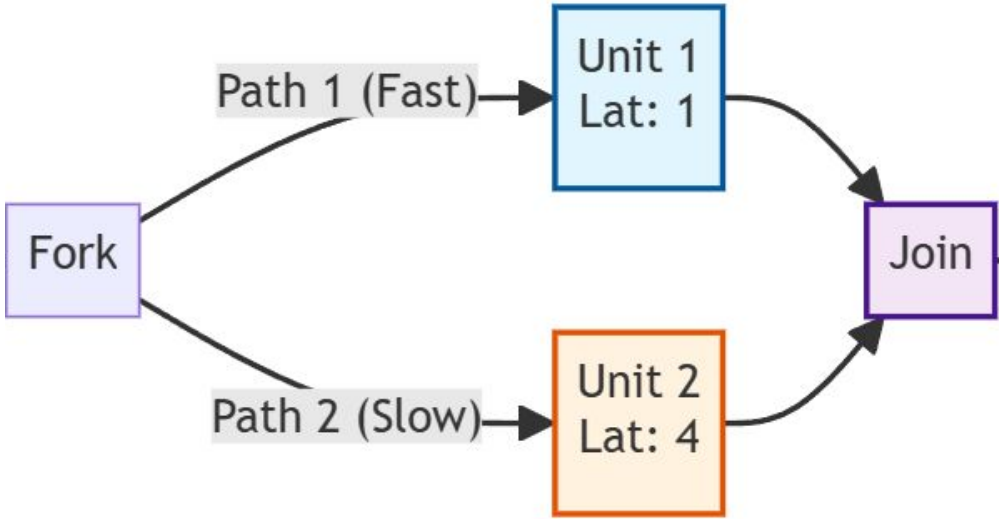
Our Solution: Just Add Latency! (But Better)

- Also reason about the maximum token occupancy imposed upon us (e.g with a loop), and add just enough buffers to handle it, while adding the needed latency to those buffers.



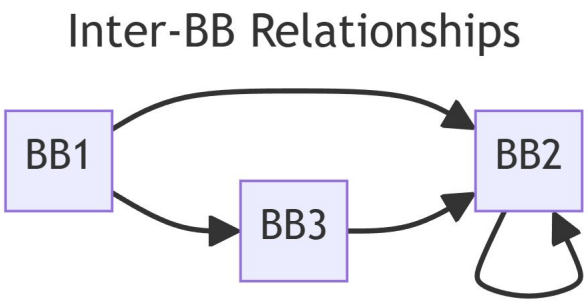
1. The Patterns in Question: Fork-Join

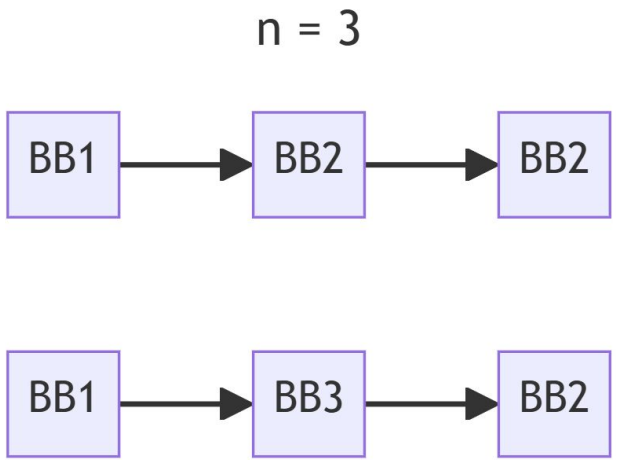
- “A fork-join pattern is the set of all reconvergent paths between a pair of a fork and a join that follow the same BB sequence.”



1. The Patterns in Question: Fork-Join

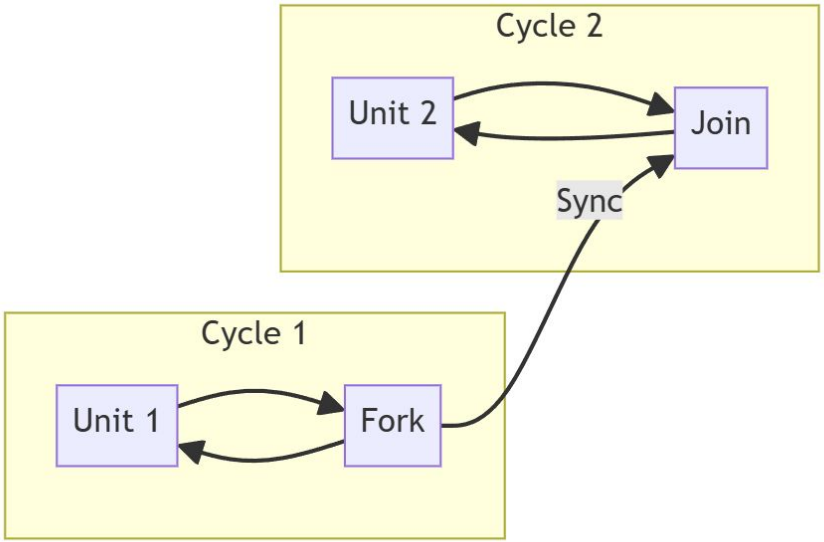
- Our approach:
  - Model the relationship between basic blocks.
  - Get all possible transition sequences of length n.
  - Enumerate reconvergent paths in those transition sequence blocks.





2. The Patterns in Question: Synchronizing Cycle Pairs

- Criteria:
  - Two cycles are disjoint
  - They belong to the same CFC
  - There's at least one join that reachable from both cycles without crossing a cycle edge.



Stage 1: Latency Balancing

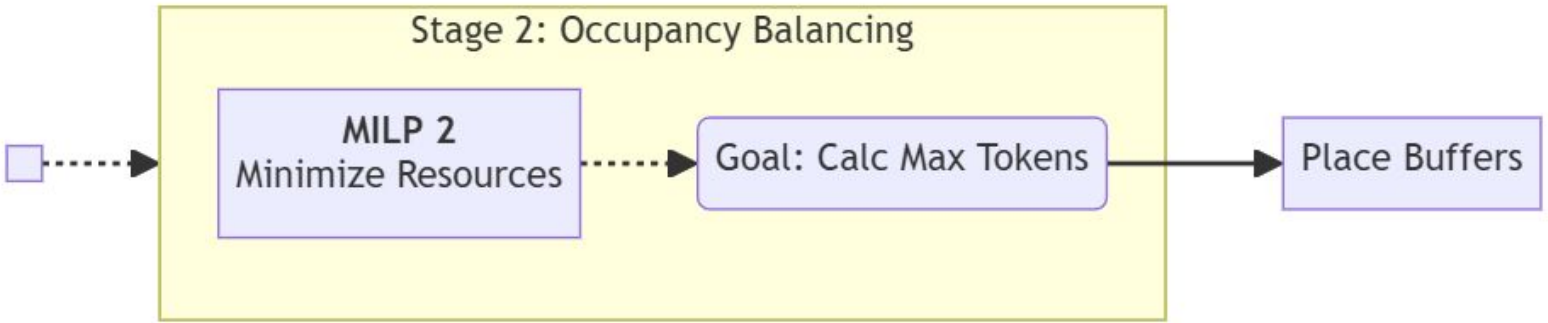
- ❏ It tries to make the latency of reconvergent paths equal.
- ❏ It is constrained by a user-defined initiation interval.
- ❏ We prioritize removing stalls, then adding latency.



Stage 2: Occupancy Balancing

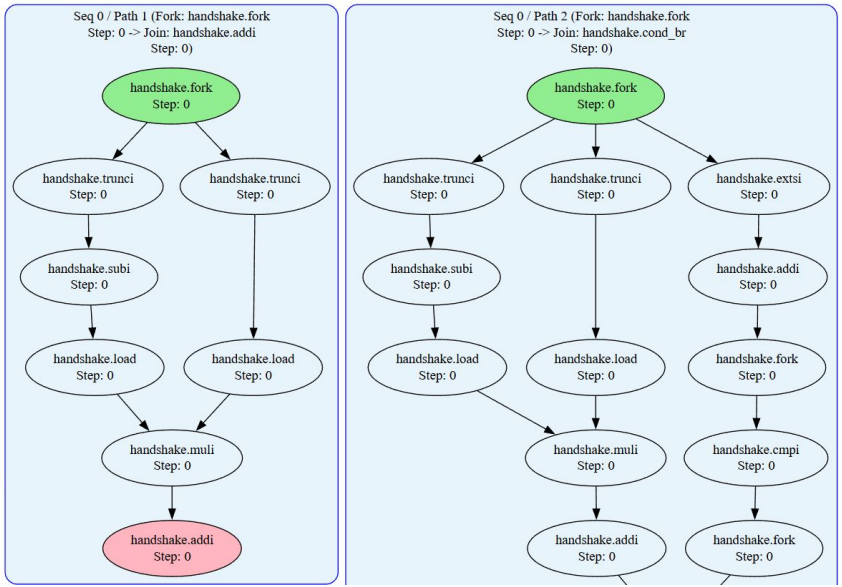
- It takes the Latency from the first step as a fixed constant.
- It calculates, the maximum number of tokens on a channel.

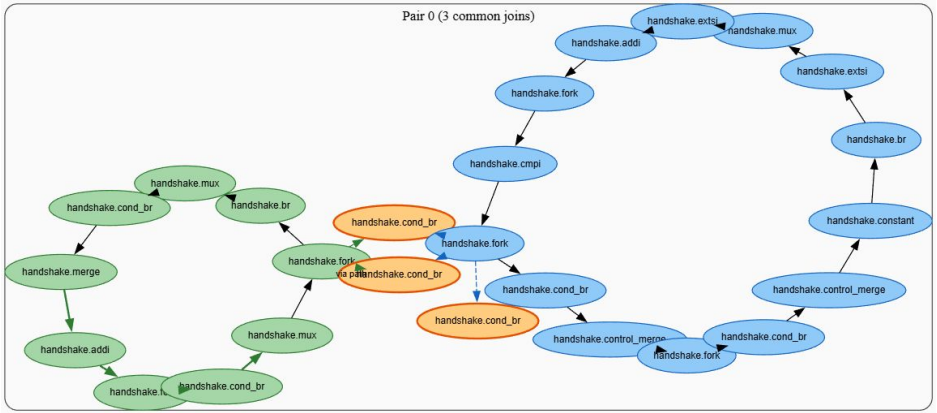
$$N_{max} \geq \frac{L^c}{II}$$



My Contribution

- Merged Pull Requests:
  - [Buffers] Reconvergent paths enumeration (#682)
  - [Buffers] Synchronizing cycle enumeration (#688)

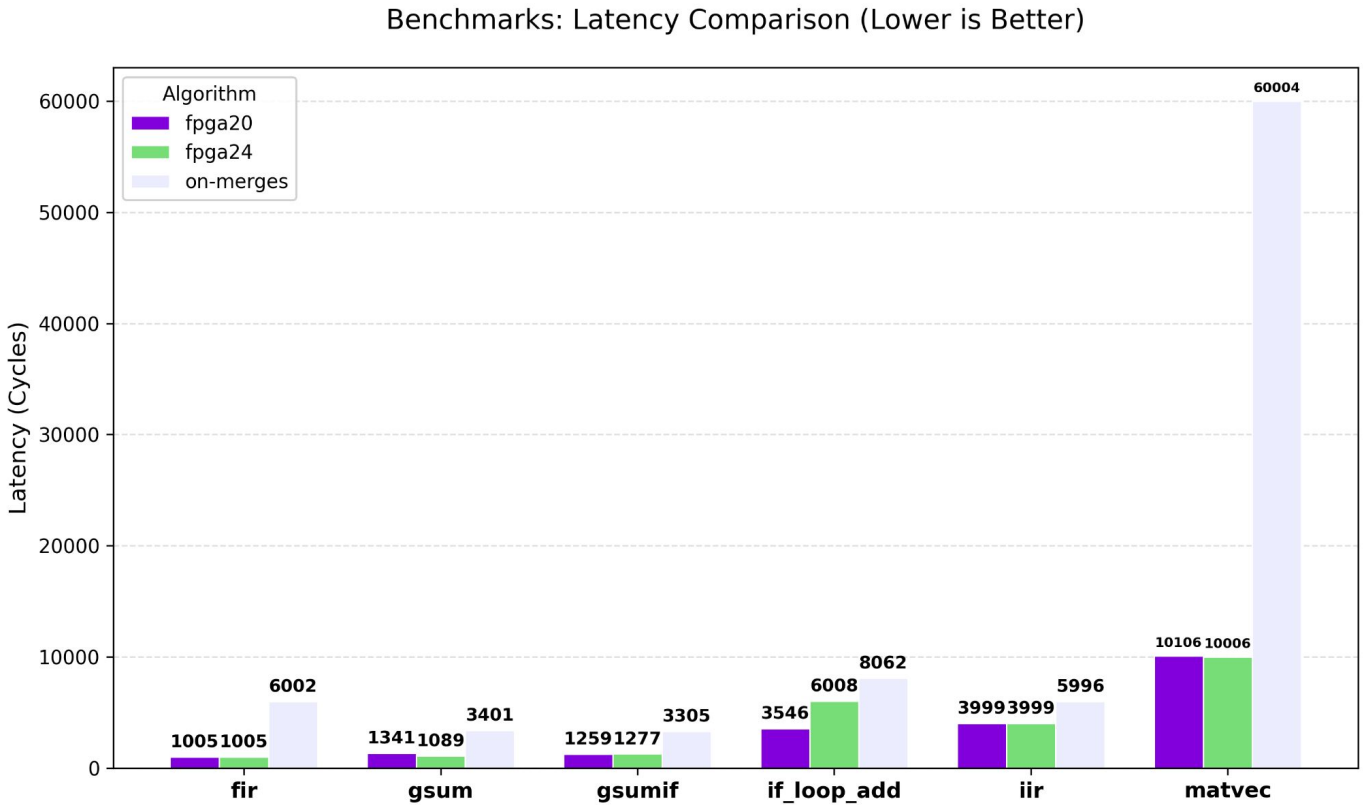




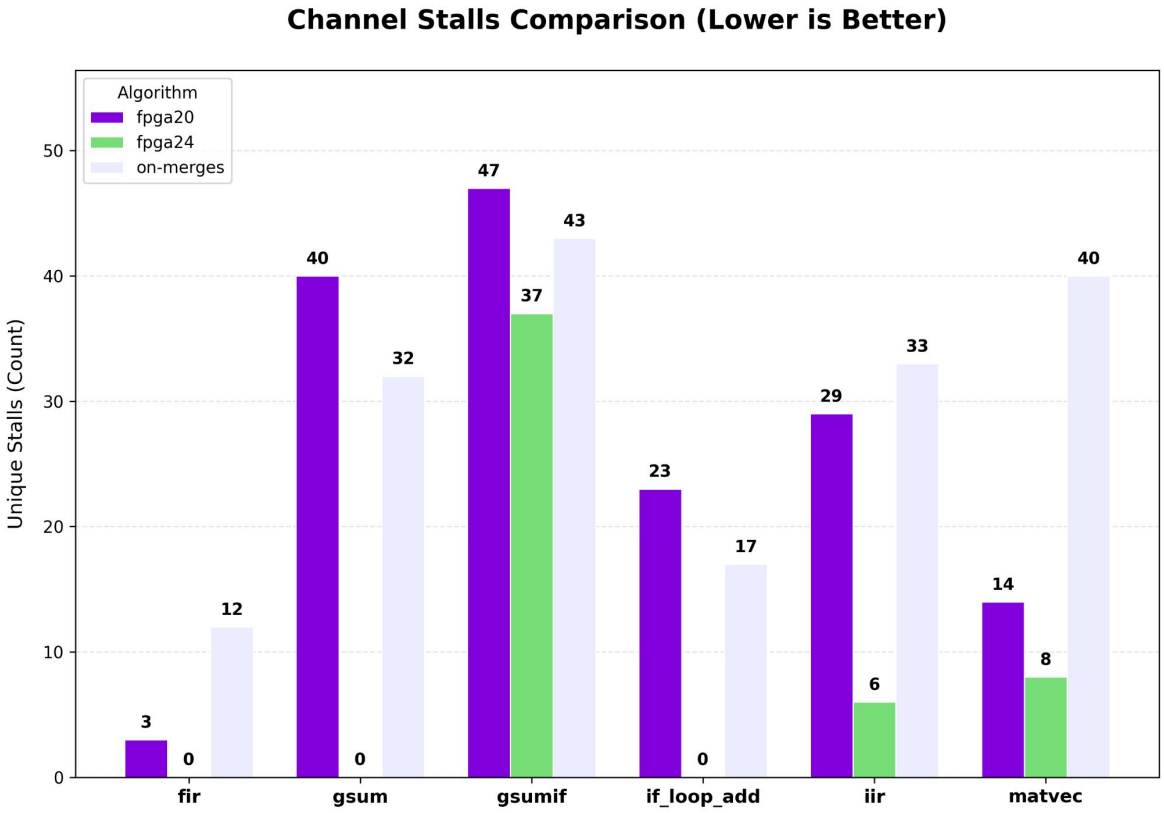
My Contribution

- Ongoing Pull Requests:
  - [Buffers] Add latency and occupancy balancing LPs (#711)

Results of My Contribution So Far:



Results of My Contribution So Far:



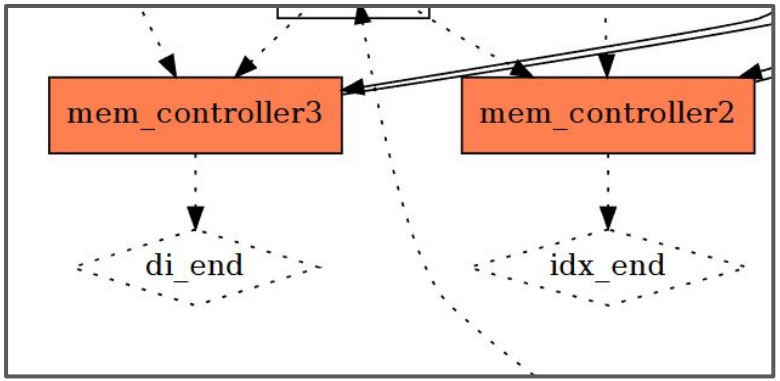
How Did I Validate My Results?

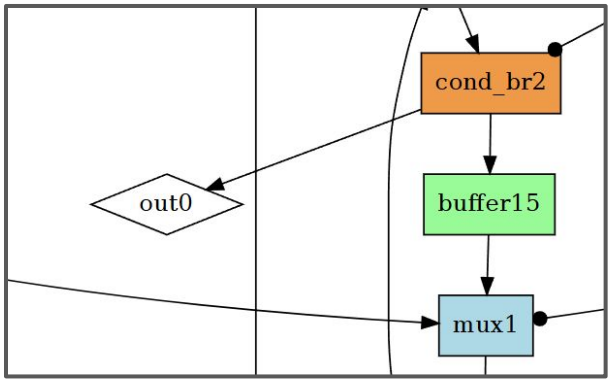
1. Comparing to FPGA20
    - If everything is implemented correctly, they should give the same performance latency-wise, unless the benchmark has irregular control flow, forcing us to take the maximum II.

2. Logging the stalls and comparing them.
    - Tracing back the stalls through a generated GraphViz graph until we find its source.

Why Are There Still Stalls?

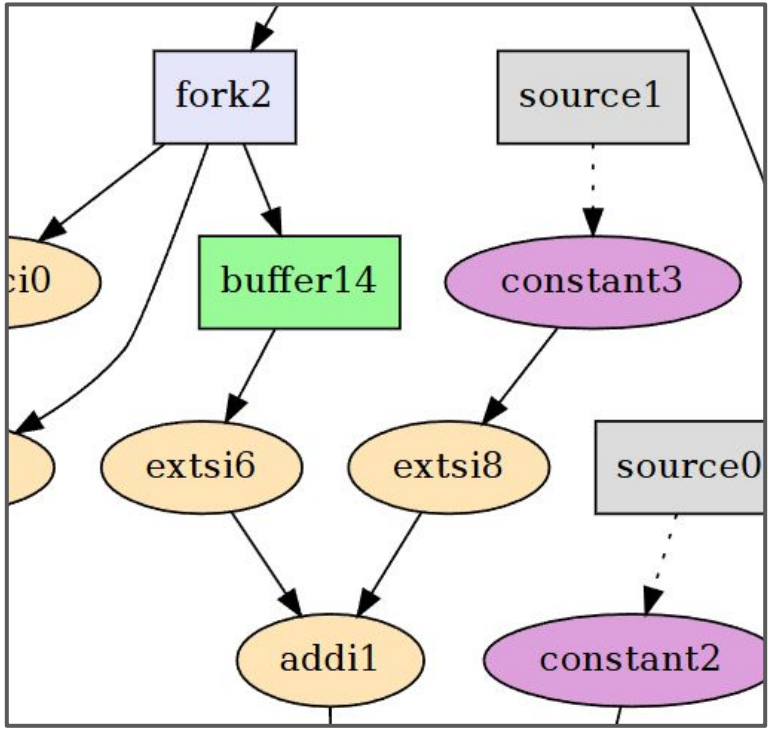
1. Some stall sources are not modeled in the graph:
    - Example: Memory controller ends and function outputs have synchronizing behavior, yet they are not modeled as a join in the input graph.
    - Solution: Add abstractions in the graph that handle these edge cases.





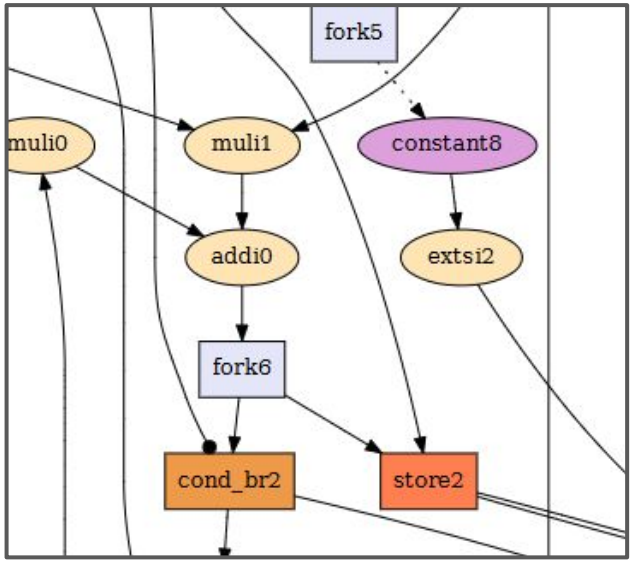
Why Are There Still Stalls?

2. Trivial stalls (driven by source / constant) are flagged as critical stalls.



Why Are There Still Stalls?

3. Circuit-Specific edge cases that need accounting for
    - Example: Loads and Stores having variable latency only if they are connected to LSQ
    - In iir, this StoreOp caused a major latency imbalance we almost did not catch:



Thank you!