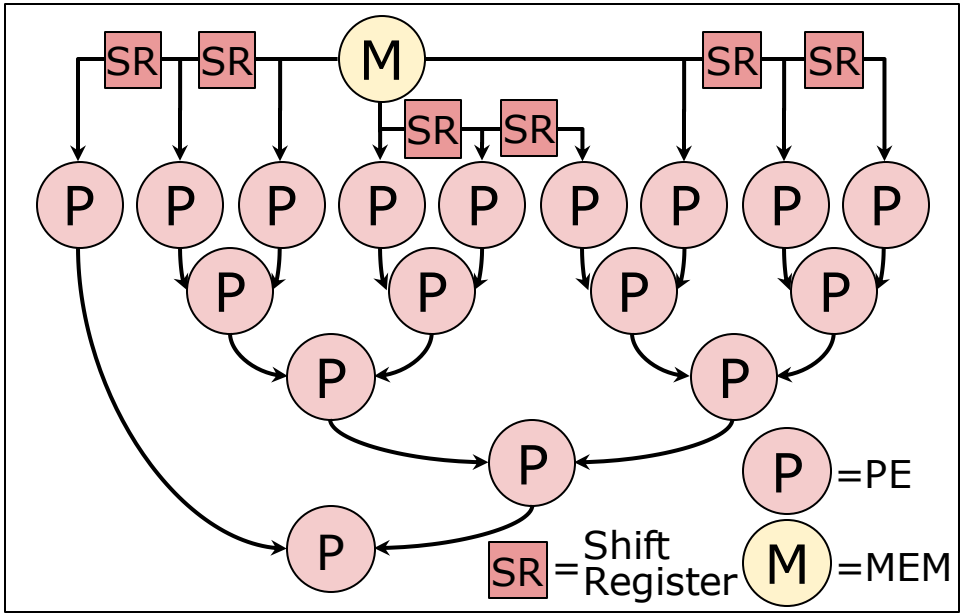
Cascade: An Application Pipelining Toolkit for Coarse-Grained Reconfigurable Arrays

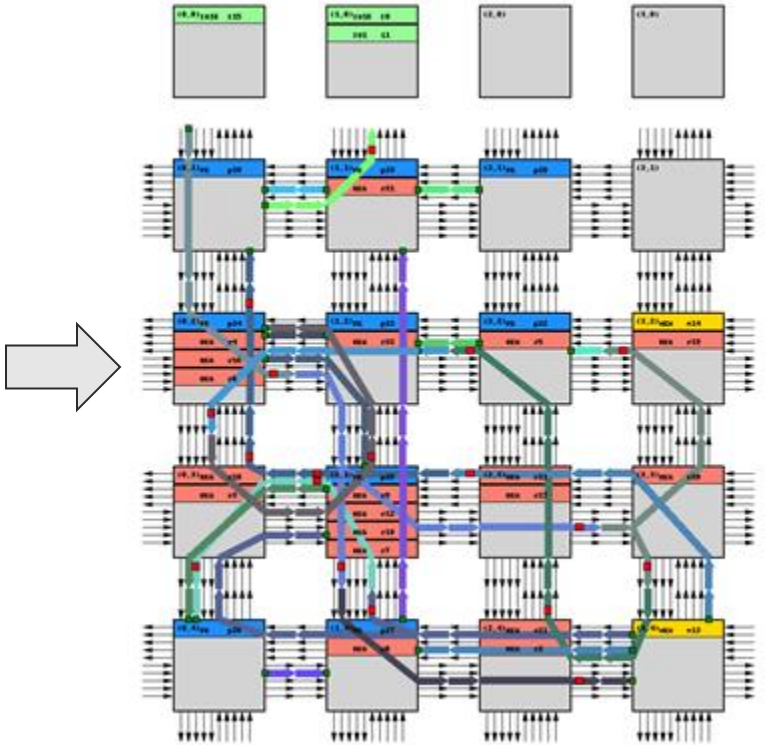
Jackson Melchert

Stanford University

Place and Route

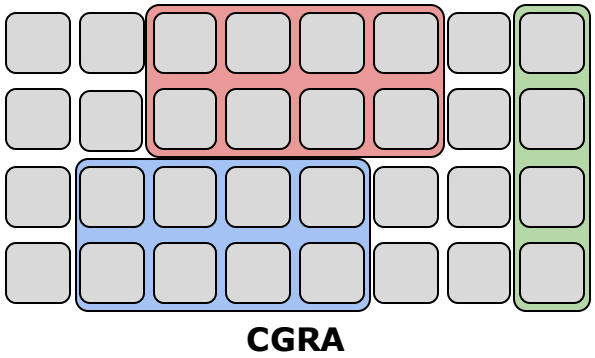
- Place and route is where abstract tiles and connections are mapped to physical locations on the array

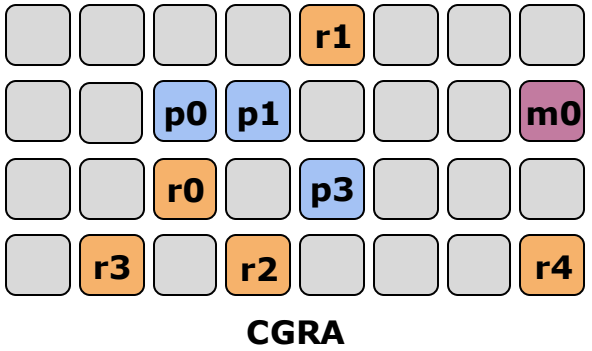




Place and Route Background - Placement

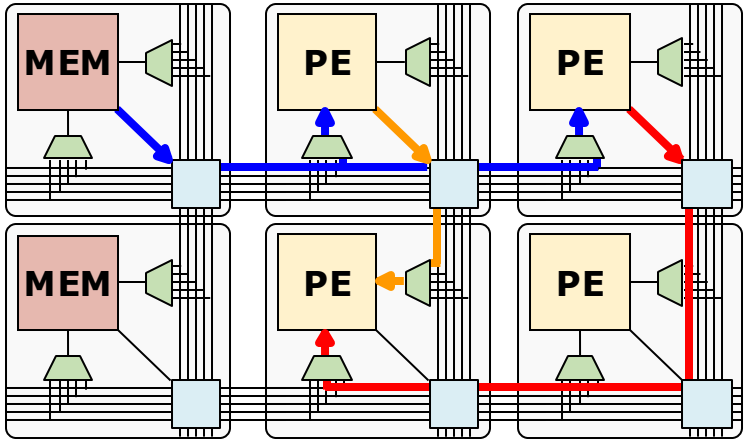
- Placement occurs in two stages:
  - Global placement:
    - Application is clustered into connected tiles
    - Clusters are given a rectangular region on the array
    - Simulated annealing used to find a non-overlapping placement of boxes
  - Detailed placement
    - Exact locations are assigned for each tile in the netlist
    - Cost function is half-perimeter wirelength (HPWL)
    - HPWL is calculated for each net and summed for the entire placement
    - Simulated annealing used to find best placement





Place and Route Background - Routing

- Routing is done using an iterative strategy based on A* path finding
- Costs factors are adjusted dynamically based on:
  - Historical usage
  - Net slack
  - Current congestion
- Routing is completed when any legal routing result is produced



Motivation for Cascade

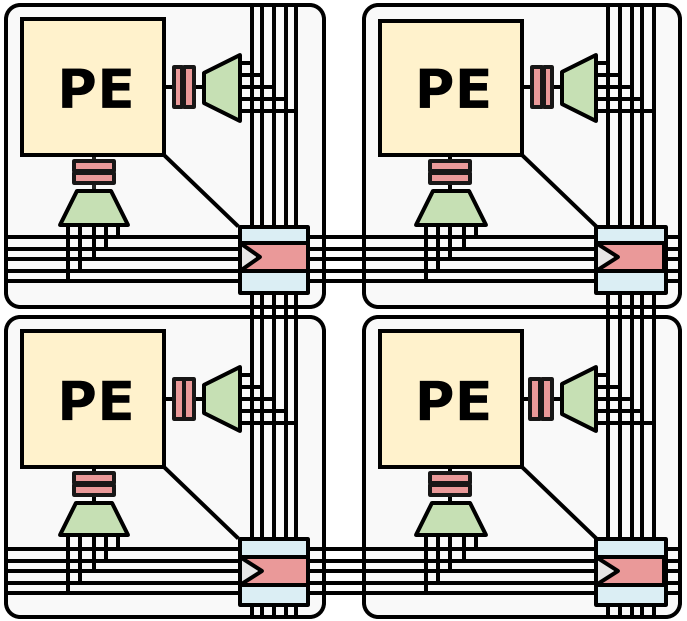
- CGRAs have large performance and energy efficiency benefits over FPGAs
  - But to achieve commercial viability, CGRAs must demonstrate performance and EDP that approaches ASICs

- Current CGRA compilers either do not pipeline resulting in low performance or pipeline exhaustively resulting in high power
  - Need a compiler with a focus on pipelining

Background - CGRA Interconnects

Several classes of CGRA interconnects exist:

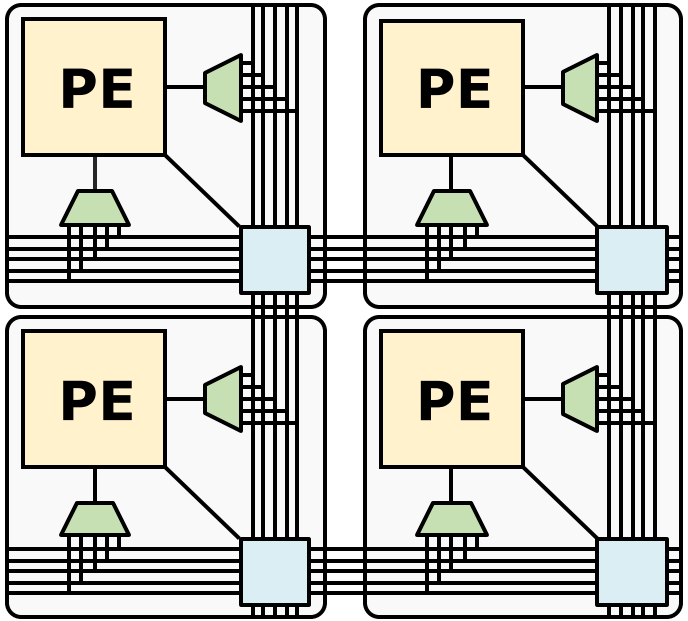
1. Exhaustively pipelined interconnects
   - Expensive



Background - CGRA Interconnects

Several classes of CGRA interconnects exist:

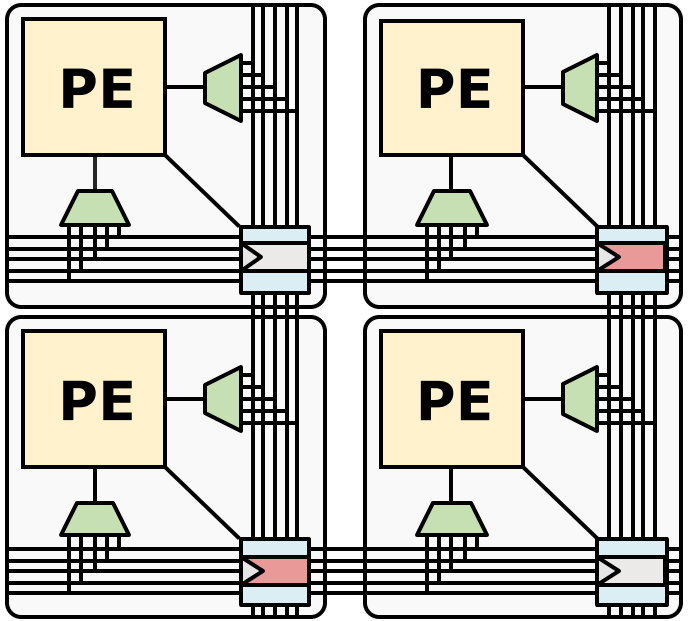
1. Exhaustively pipelined interconnects
   - Expensive
2. Non-pipelined interconnects
   - Cannot run at high frequencies



Background - CGRA Interconnects

Several classes of CGRA interconnects exist:

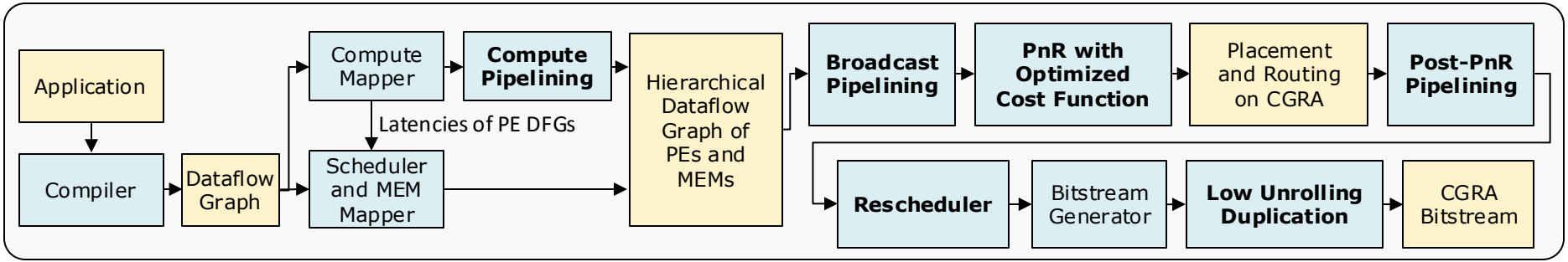
1. Exhaustively pipelined interconnects
   - Expensive
2. Non-pipelined interconnects
   - Cannot run at high frequencies
3. Interconnects with configurable registers
   - Need pipelining techniques



Key Challenges and Our Solutions

Challenge 1	**Challenge 2**	**Challenge 3**
Existing CGRA compilers don't pipeline applications effectively	Too many register resources get used during pipelining	Need a way to estimate the clock frequency of the application on evolving CGRAs
A variety of existing and novel pipelining techniques, including post-PnR pipelining	A technique for absorbing non-critical path registers into memory tiles	An automatic CGRA timing model generator and a static timing analysis tool for CGRA applications

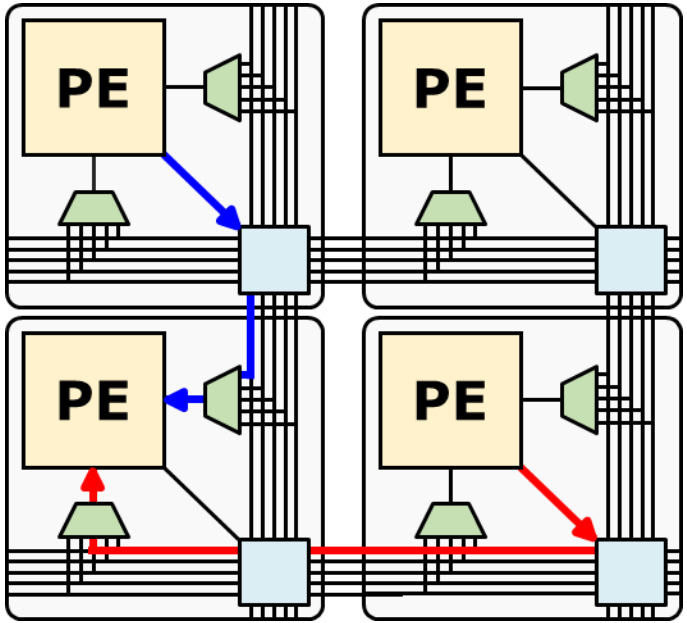
Cascade



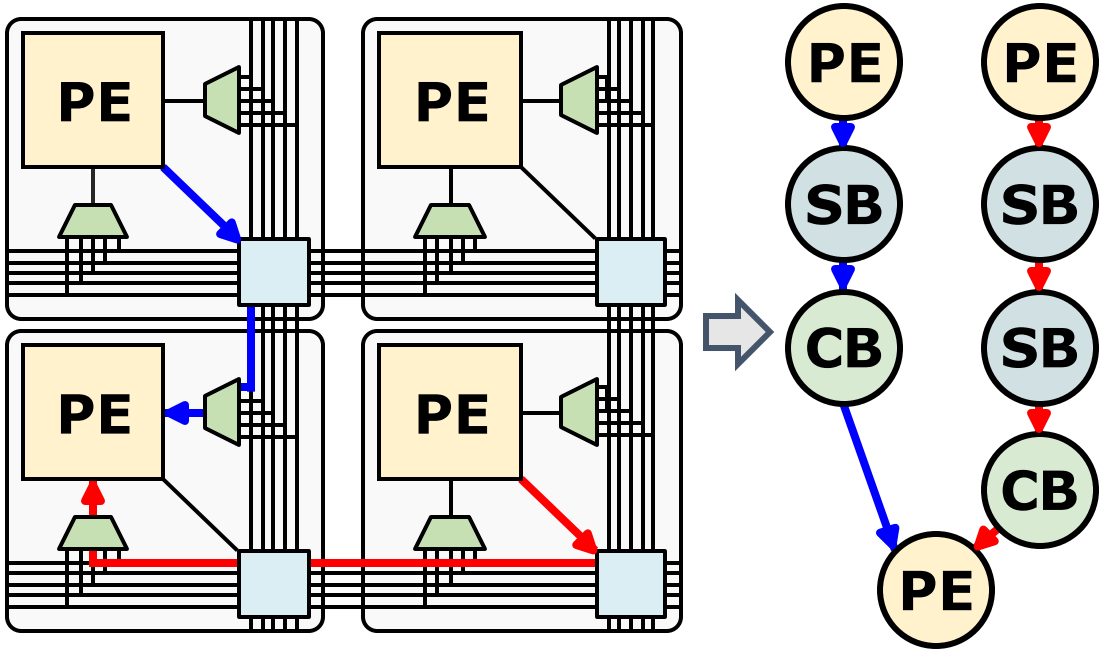
- Cascade includes:
  - An automatic CGRA timing model generator
  - A static timing analysis tool for CGRA applications
  - Multiple stages of pipelining, including post-PnR pipelining to achieve high performance
  - Register absorption for reducing resource consumption

Post-Place and Route Pipelining

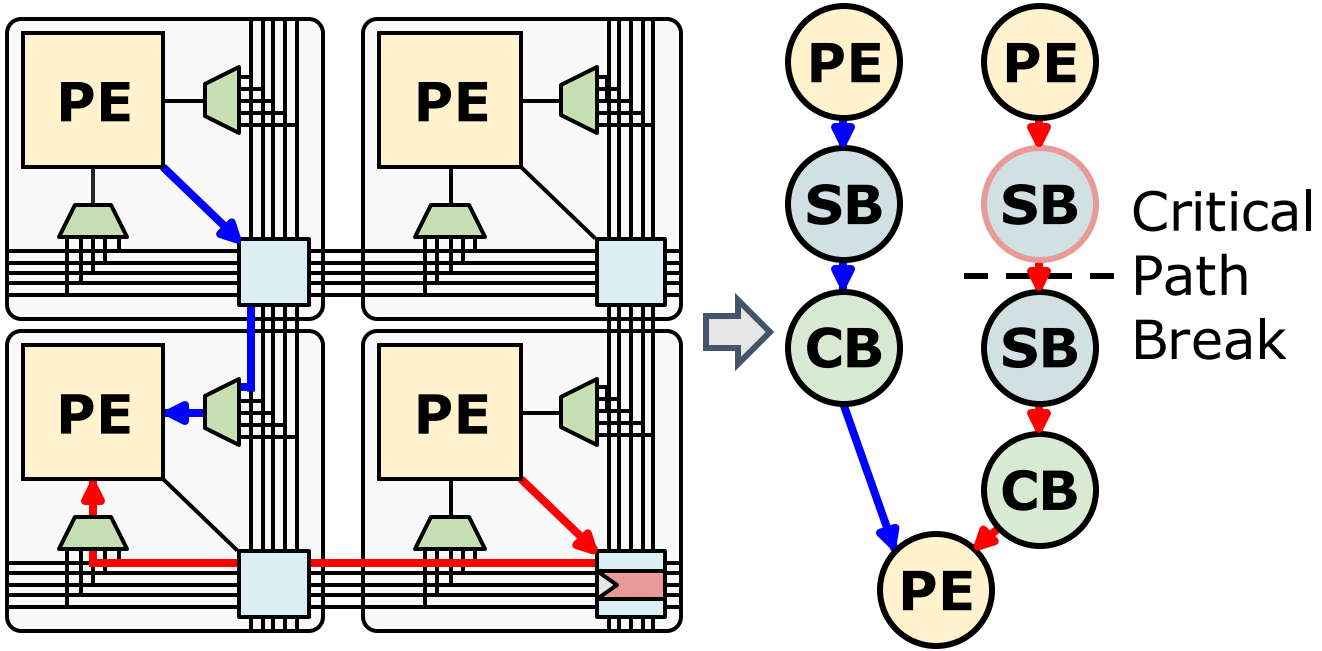
- After place and route, we know locations of all tiles and nets
- We iteratively identity the critical path, break it, reschedule the application, and repeat
- We can only add registers to existing routes, so eventually we run out of pipelining opportunities



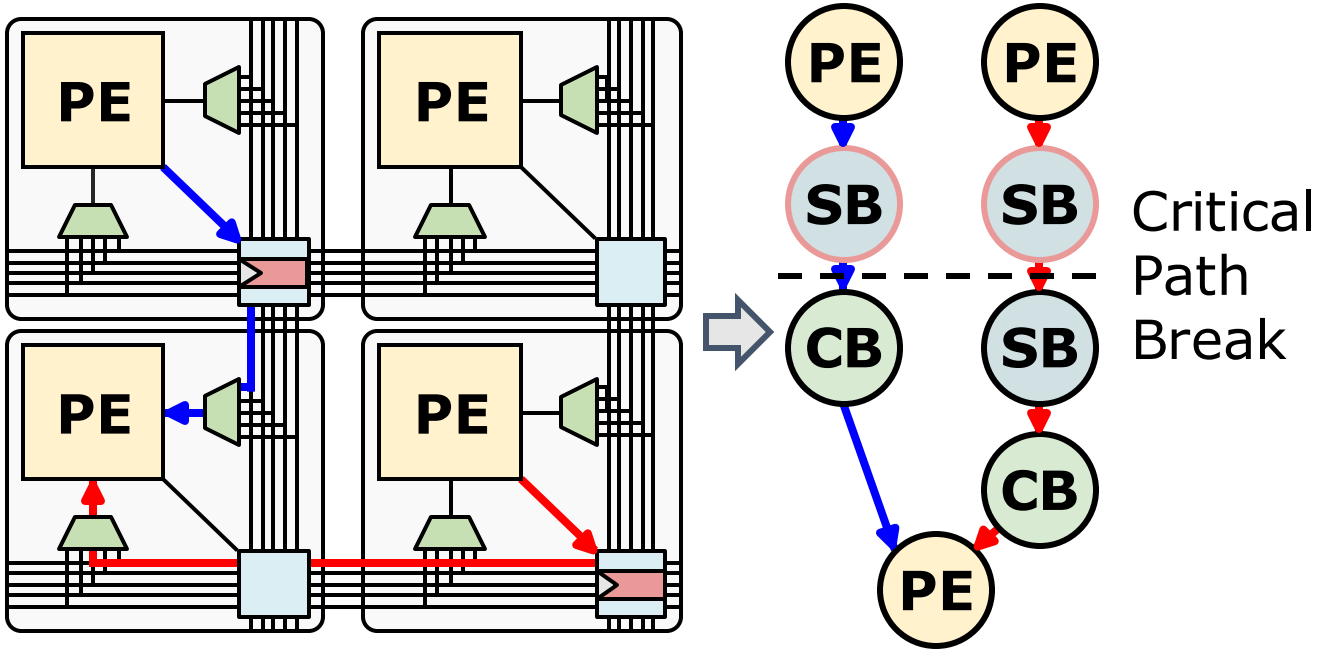
Post-Place and Route Pipelining Example



Post-Place and Route Pipelining Example



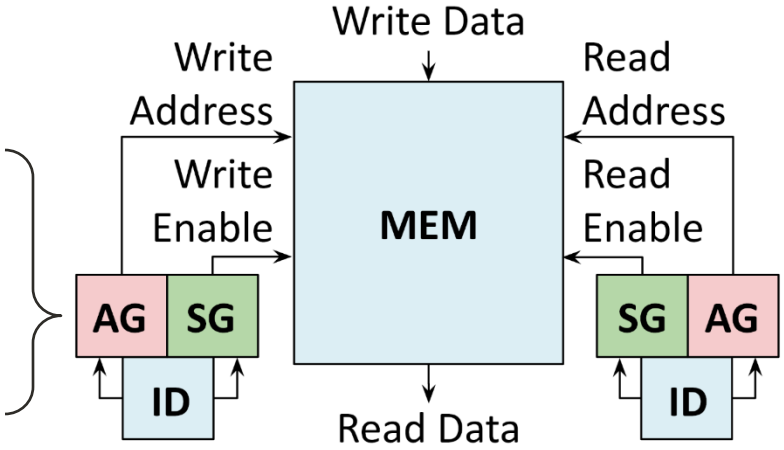
Post-Place and Route Pipelining Example



Rescheduling CGRA Applications

- After analyzing the new compute kernel latencies, **we only update the configuration registers in the MEM tiles that control scheduling** to maintain application functionality
  - Avoid changing the application DFG or placement or routing

```
“cycle_starting_addr”:[5]
“cycle_stride”:[1,5]
“dimensionality”:2
“extent”:[2,5]
```



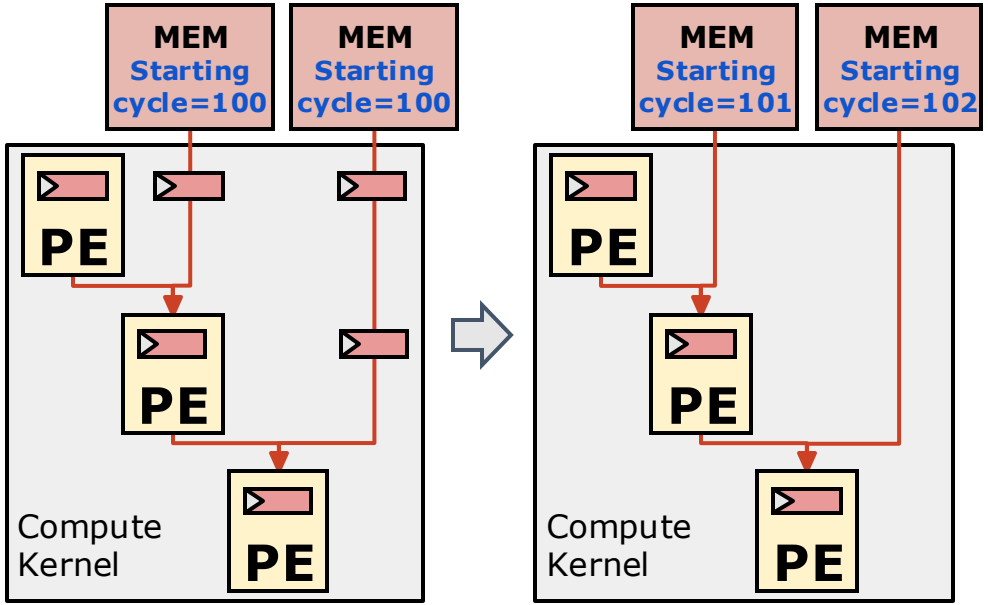
Rescheduling CGRA Applications

1. Statically schedule the application without registers
2. Place and route the application
3. Pipeline
4. Analyze the application and compute new latencies
5. Incrementally reschedule the application by updating the memory tile configuration registers

Avoids cyclic scheduling, PnR, and pipelining

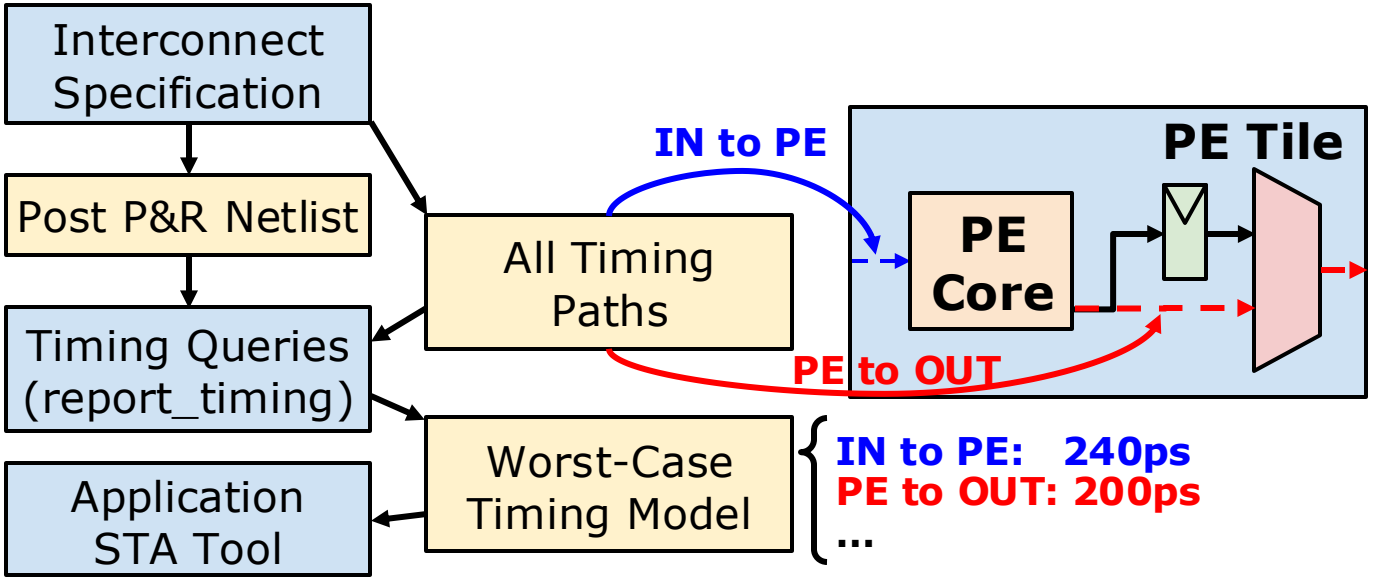
Optimizing Register Resource Usage

- Registers that are **not on the critical path** can be removed to save energy
  - Static schedules of memory tiles can be adjusted to absorb the delays



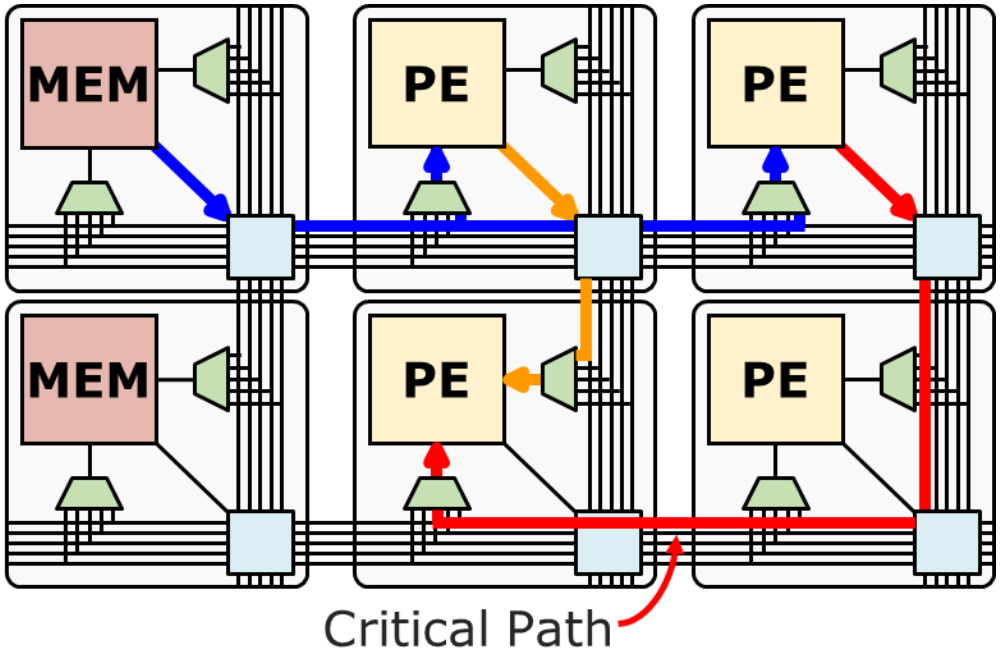
Methodology for Generating Timing Model

- Automated method for generating timing model from an interconnect specification

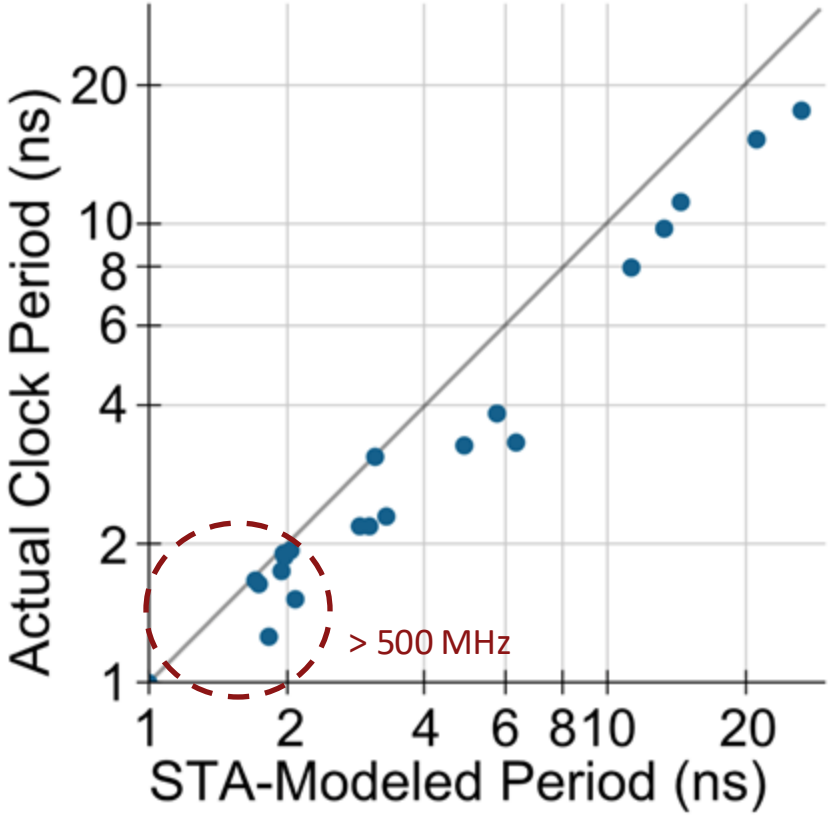


Static Timing Analysis Model

- STA model allows for determining the estimated frequency of the application running on the hardware

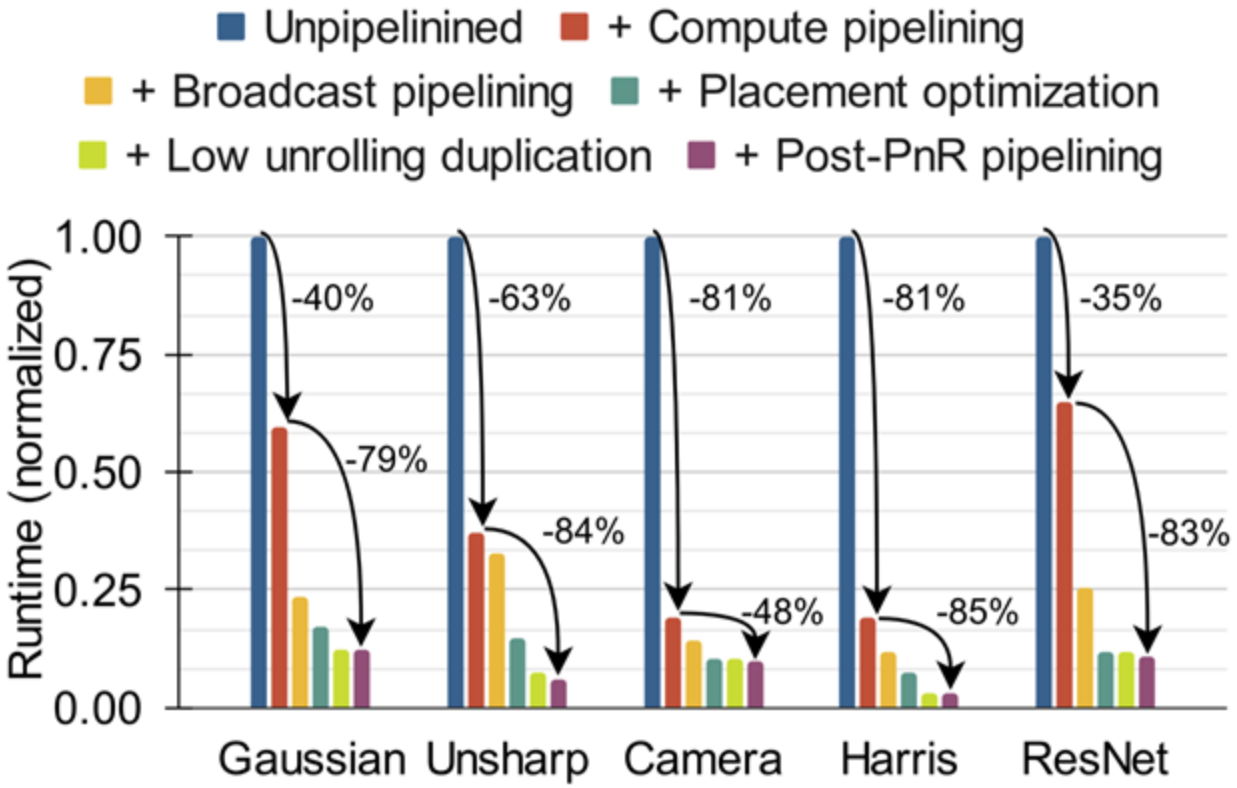


Results - STA Model Evaluation

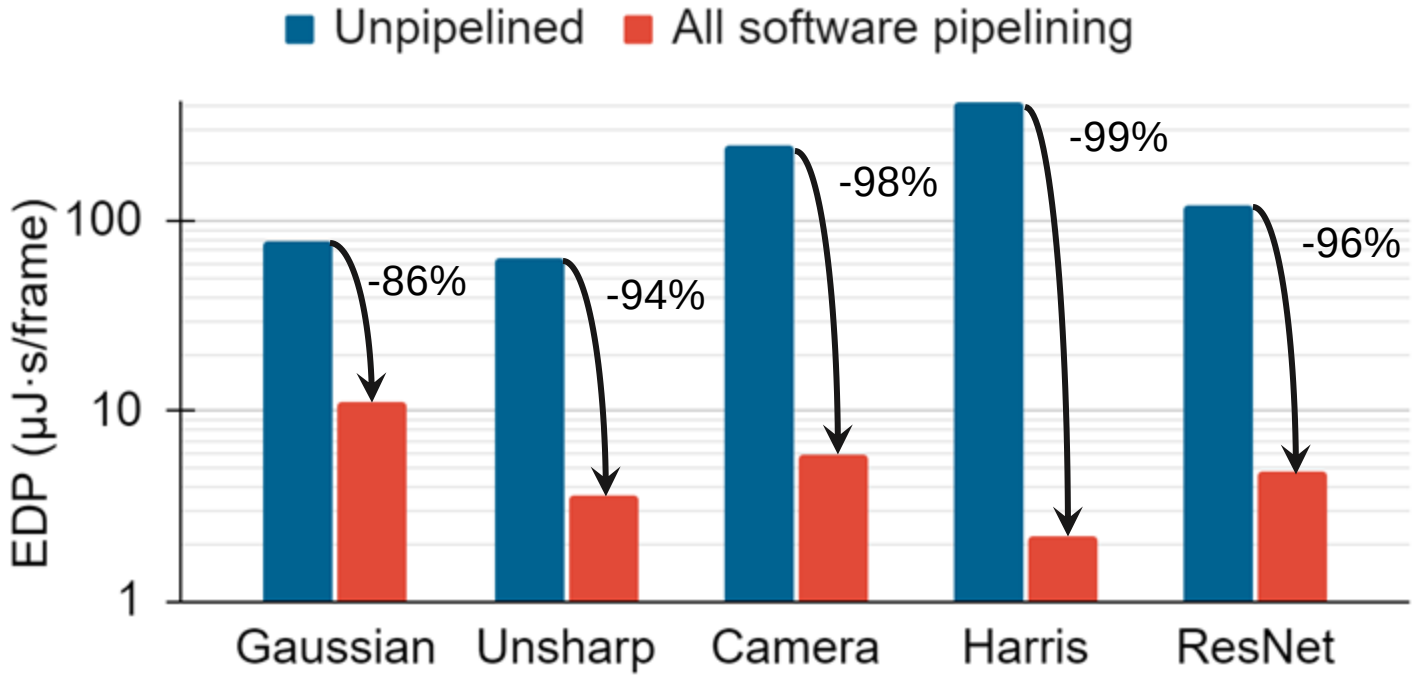


- STA model predicts the actual clock period accurately
  - Above 500 MHz, the average error is 13%
- Actual clock period is lower than the STA-modeled period
  - Our model is pessimistic, it will provide a lower bound for the clock frequency

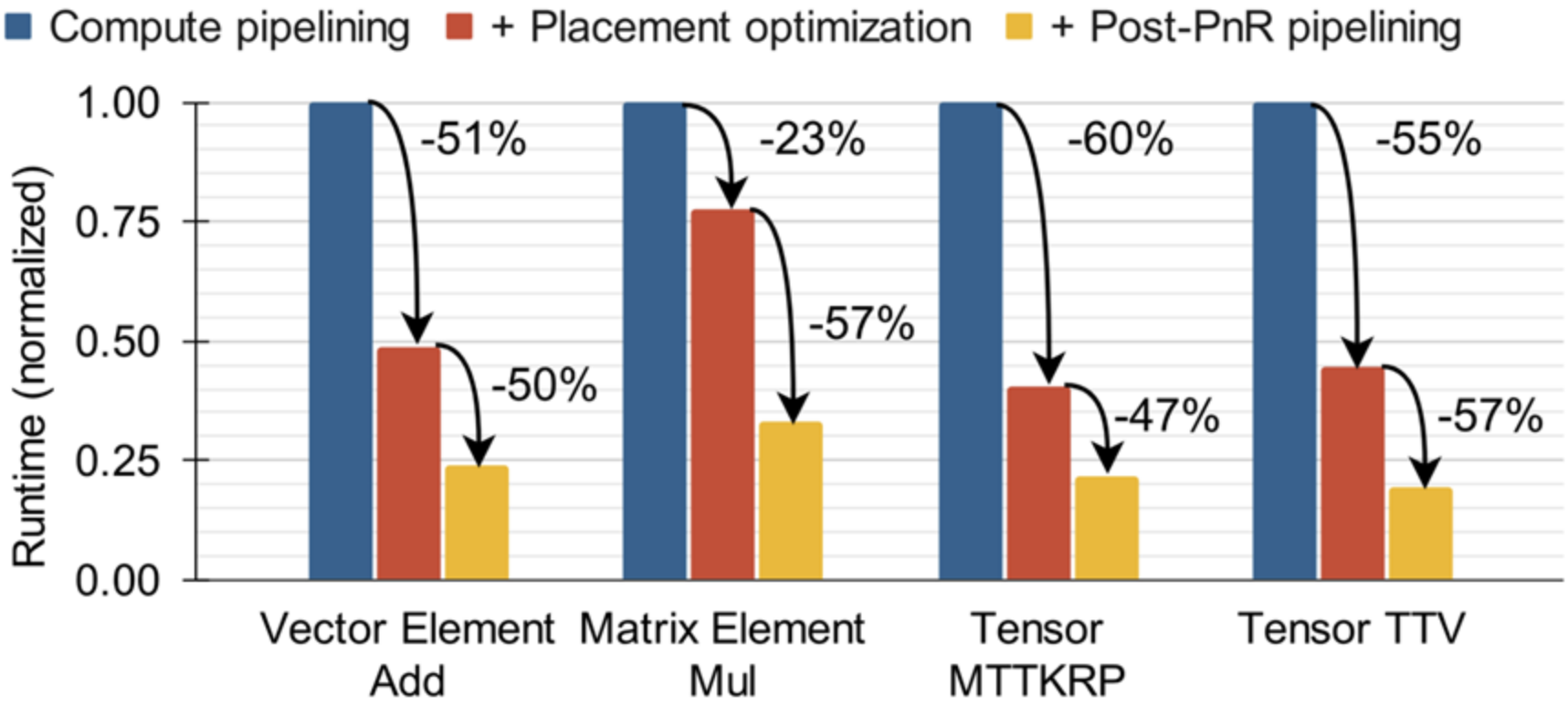
Effect of Pipelining on Dense Application Runtime



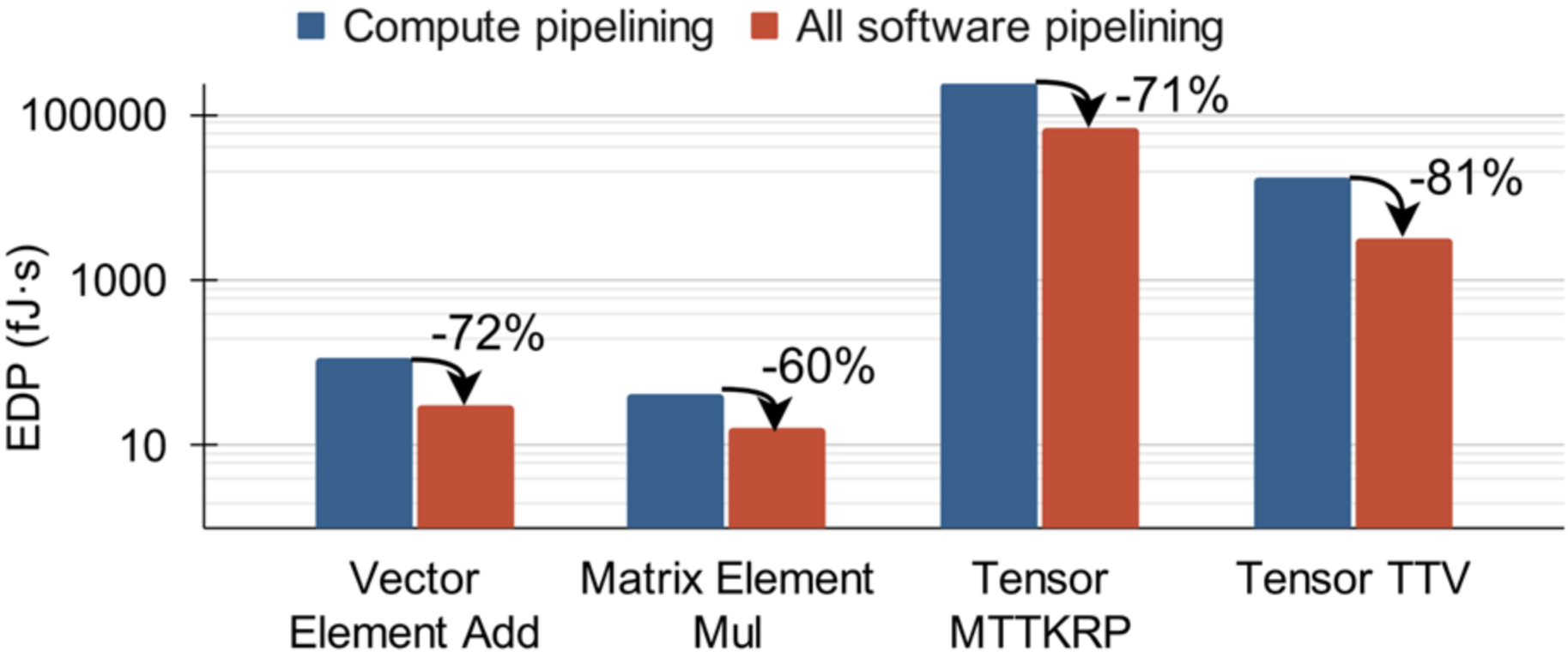
Effect of Pipelining on Dense Application EDP



Effect of Pipelining on Sparse Application Runtime



Effect of Pipelining on Sparse Application EDP



Demo

- We'll use the same application as the SAM demo
- To run PnR without post-PnR pipelining use the following command:
- `./cascade_demo.sh 0`
- This will produce a visualization of the PnR result with the critical path highlighted in /aha/garnet/SIM_DIR/pnr_result_17.png
- Next, we'll turn on post-PnR pipelining
- `./cascade_demo.sh max`

Cascade Summary

- Cascade is an open-source end-to-end CGRA compiler that achieves high performance and low EDP
- We adapt prior work on pipelining to CGRAs and introduce novel post-PnR pipelining and register absorption techniques
- We create a methodology for generating timing models for evolving CGRAs
- Cascade achieves
  - 8 - 34× shorter critical paths and 7 - 190× lower EDP for dense applications
  - 3 - 5.2× shorter critical paths and 2.5 - 5.2× lower EDP for sparse applications