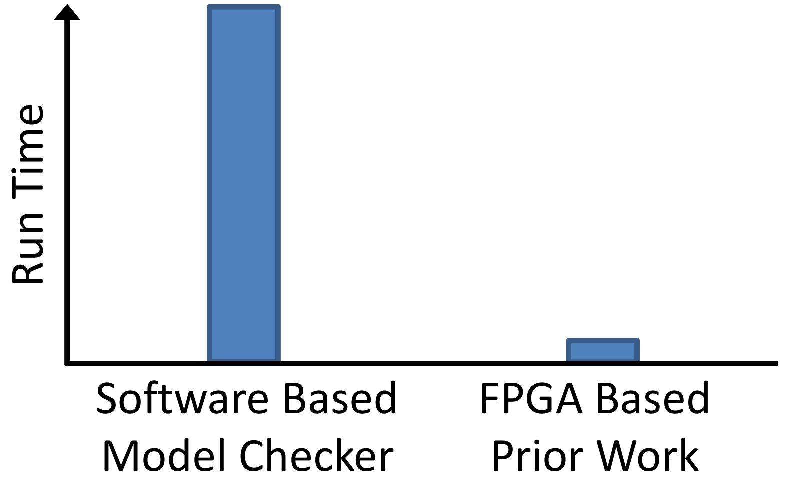
Runtime Programmable Pipelines for Model Checkers on FPGAs

Mrunal Patel, ***Shenghsun Cho***, Michael Ferdman, Peter Milder

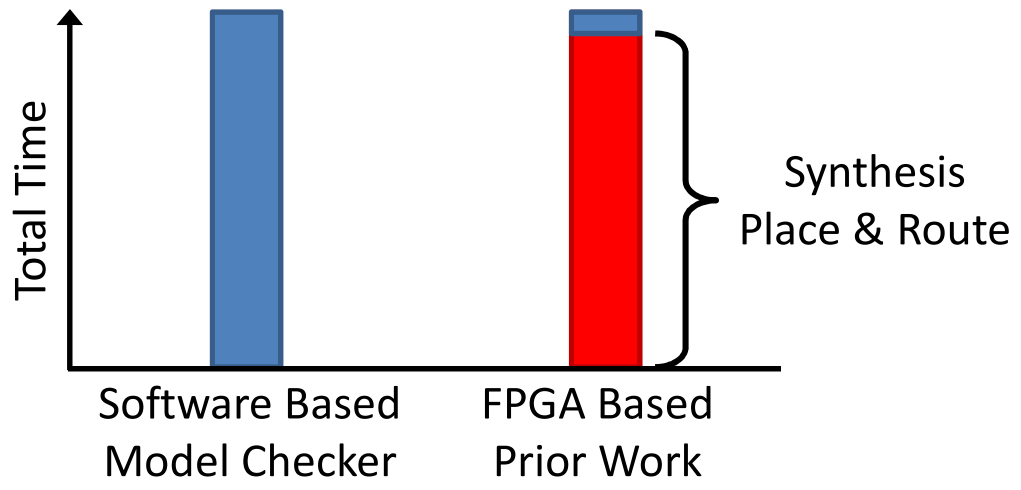
FPGA Accelerates Model Checking

- Model checking ensures system correctness
  - By exploring all states of a system
  - Hours or days of run time with general purpose cores
- FPGAs show orders-of-magnitude speedup in run time
  - 3 hours → 12 seconds reported by prior work [Cho'18]



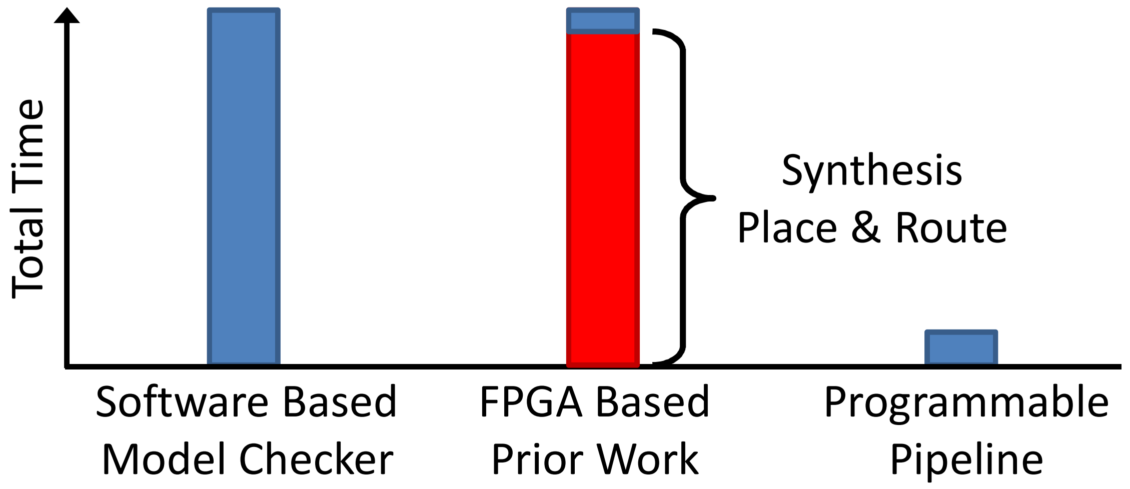
FPGA Preparation Time Voids The Benefit

- Models change while system development in progress
  - Fixing bugs, adding features, etc.
- Every model change result in new hardware logic
  - Requires hours to prepare (syn, P&R) new model checkers
- Prevents rapid system development iteration



Our Approach: Programmable Pipeline

- Instruction controlled model checker on FPGAs
- Eliminate preparation time
  - No FPGA synthesis and P&R for new/modified models
- Limited overhead when comparing to prior work
  - Maintain 80% - 90% performance in run time



Stony Brook University

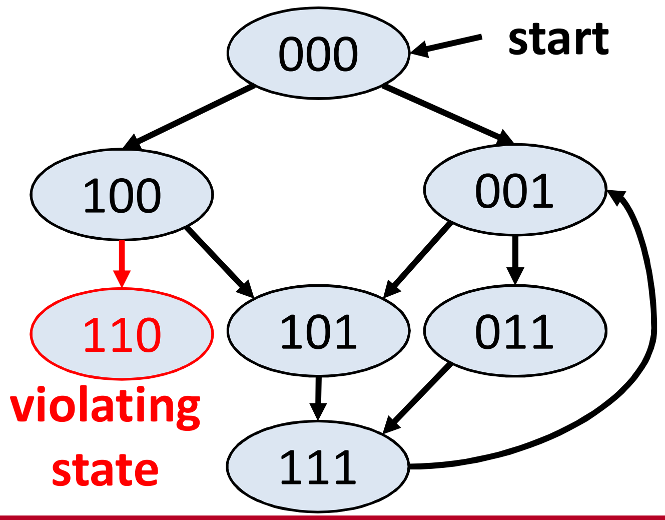
COMPAS

Outline

- Overview
- **FPGA based model checking**
- Programmable Pipeline for FPGA Model Checkers
- Evaluation
- Conclusions

Explicit State Model Checking

- Bit vector explicitly represents system state
  - Contains PC, registers, variable values, etc.
- State space is logically represented as a graph
  - Edges represent all possible transitions of system states
  - Some states represent spec violations (code assertions)
- Model checker explores the graph
  - Visits all successors of each node
    (e.g., breadth-first traversal)
  - Discovers violating states

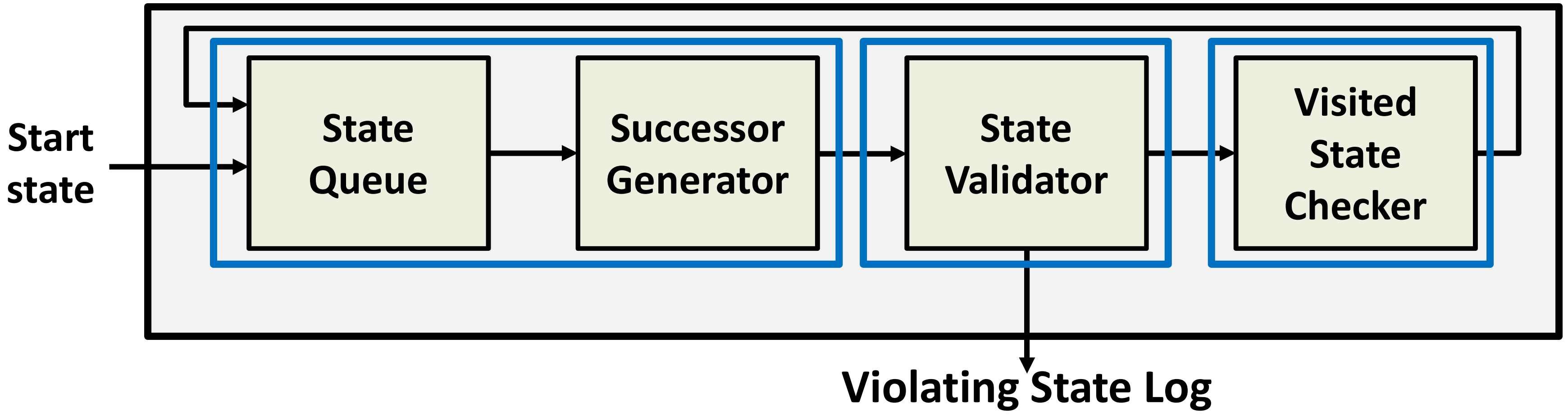


Stony Brook University

COMPAS

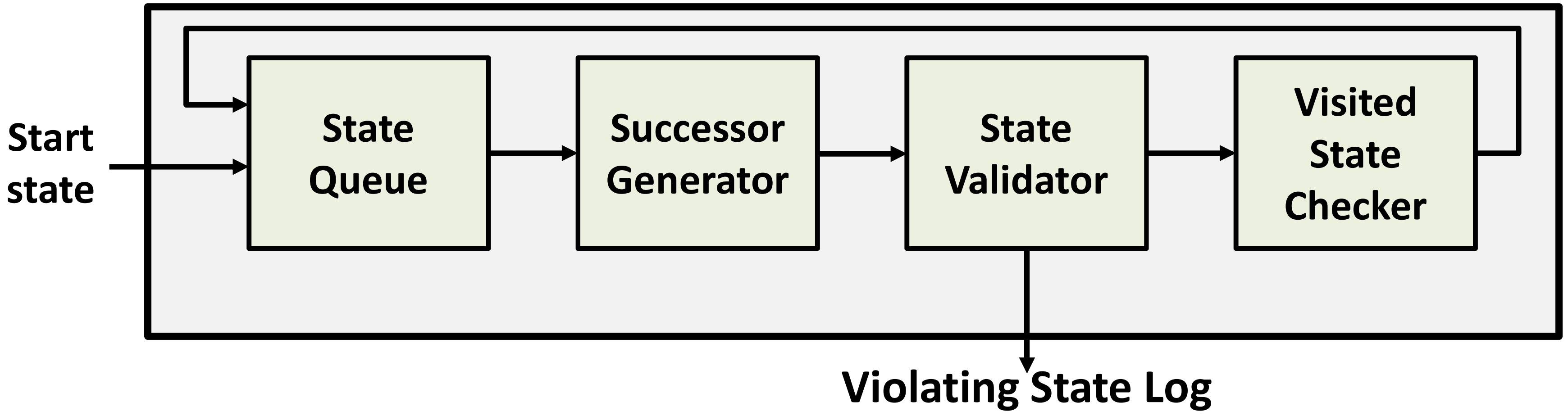
Model Checker Overview

- State space exploration:
  1. Take a state from queue, generate all successors
  2. Check and log violating states
  3. If successors not visited before, enqueue them
- Explore until queue is empty



Model Checking Challenges

- Costly computation for general purpose cores
  - Bit manipulation, memory compare, hashing, etc.
- Limited parallelism
  - Shared state queue and visited-state storage

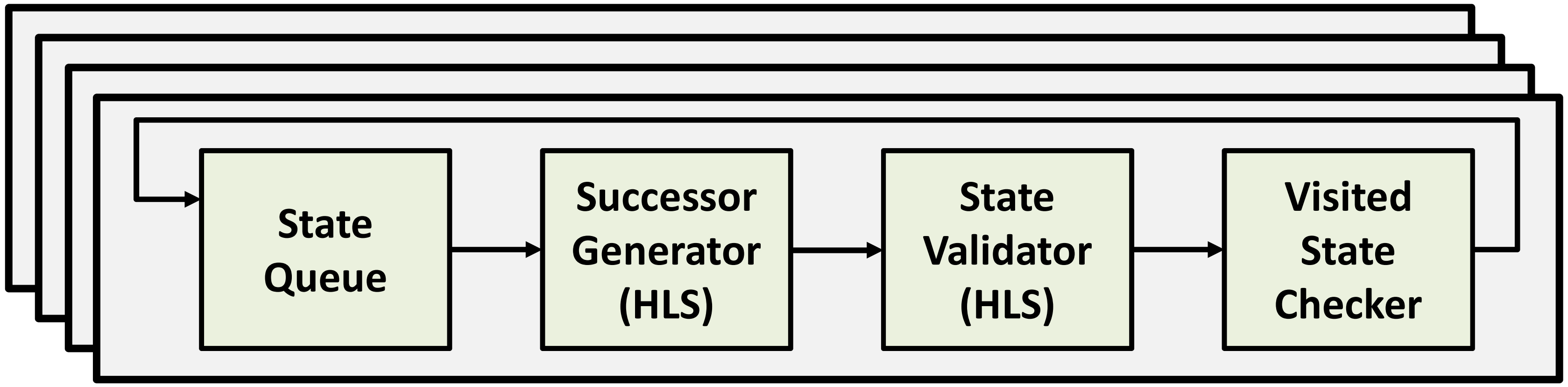


Stony Brook University

COMPAS

Model Checking on FPGAs

- FPGA dedicated logic accelerates the computation
- Independent resources enables parallelism
  - Performance grows linearly with num of model checker cores
  - Limited by FPGA BRAM capacity and BRAM usage per core
- Prior work shows significant speedup in run time

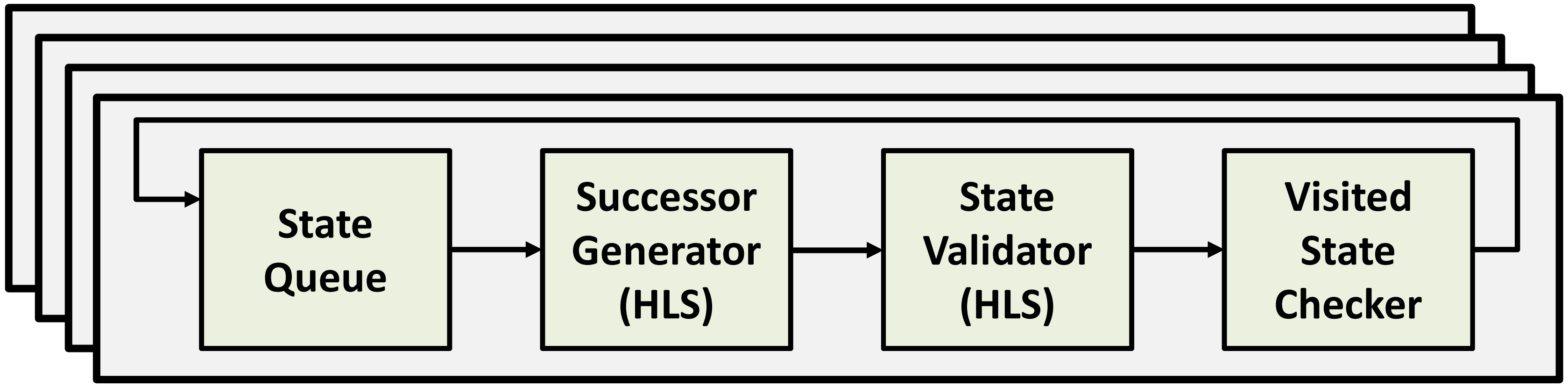


Stony Brook University

COMPAS

Model Checking on FPGAs

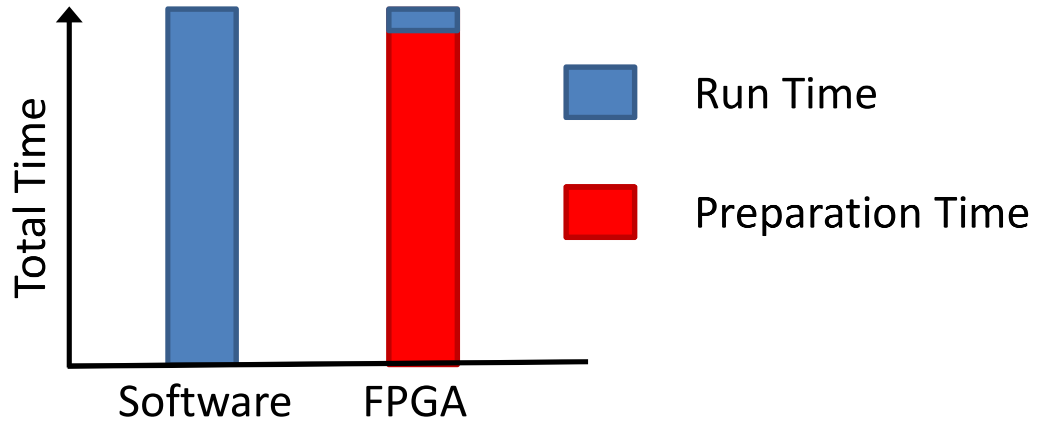
- FPGA dedicated logic accelerates the computation
- Independent resources enables parallelism
  - Performance grows linearly with num of model checker cores
  - Limited by FPGA BRAM capacity and BRAM usage per core
- Prior work shows significant speedup in **run time**



Speedup in run time does not mean overall speedup

FPGA "Preparation Time" Problem

- HLS directly translates models into hardware
  - Every model change generates new hardware circuit
  - Synthesis and P&R for every new/modified model
- High resource utilization causes long P&R
  - Hours of waiting to generate the bitstream
  - Multiple iterations for timing closure



Preparation time kills the run time speedup

Outline

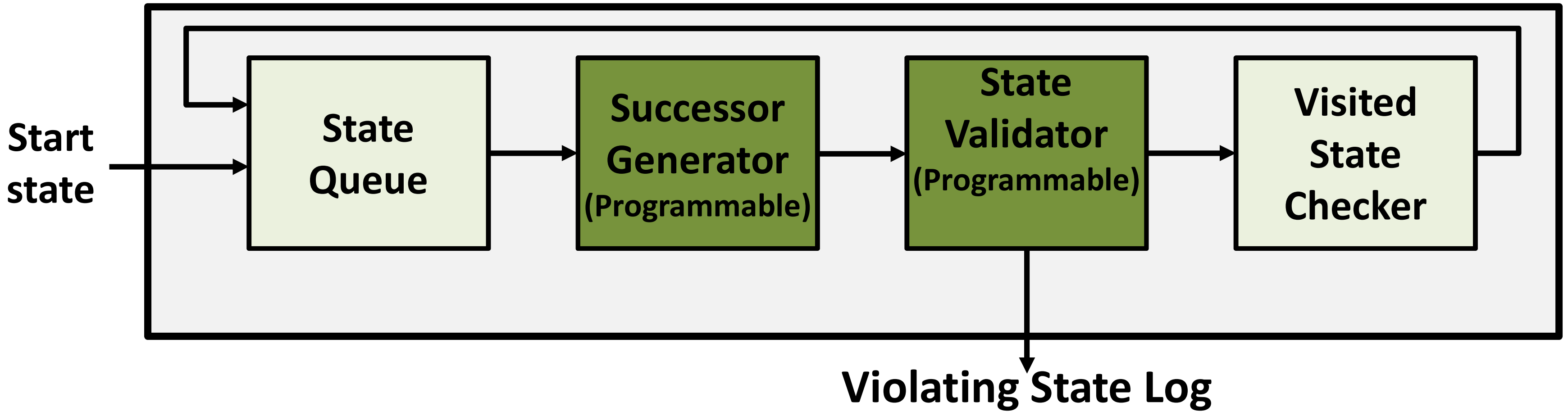
- Overview
- FPGA based model checking
- **Programmable Pipeline for FPGA Model Checkers**
- Evaluation
- Conclusions

Stony Brook University

COMPAS

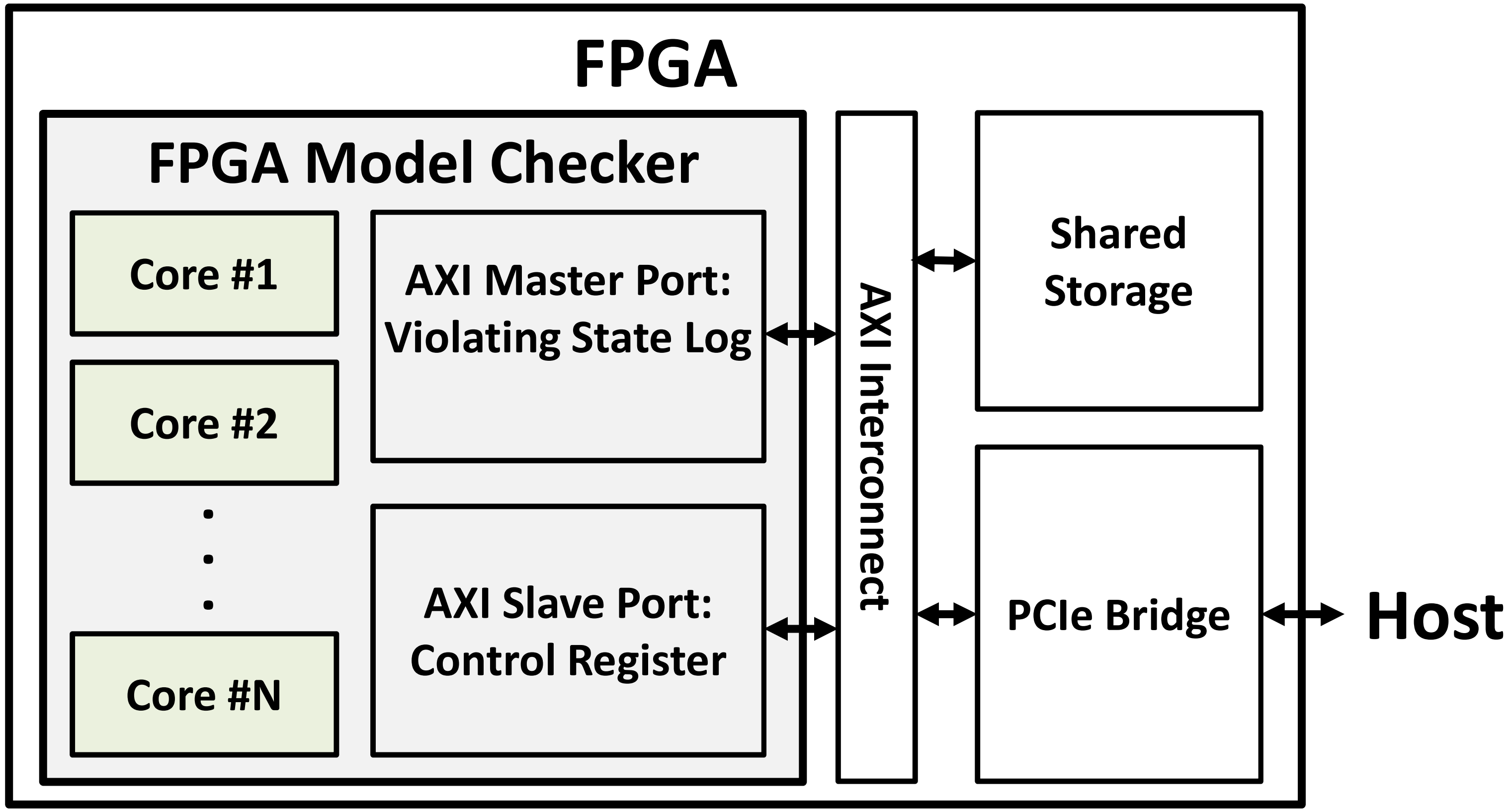
Replacing Model-Specific Logic

- Programmable pipelines replace model-specific logic
  - Successor State Generator
  - State Validator
- Maintain the same throughput as model-specific logic


Start state
State Queue
Successor Generator (Programmable)
State Validator (Programmable)
Visited State Checker
Violating State Log

Multi-Core for Parallelism

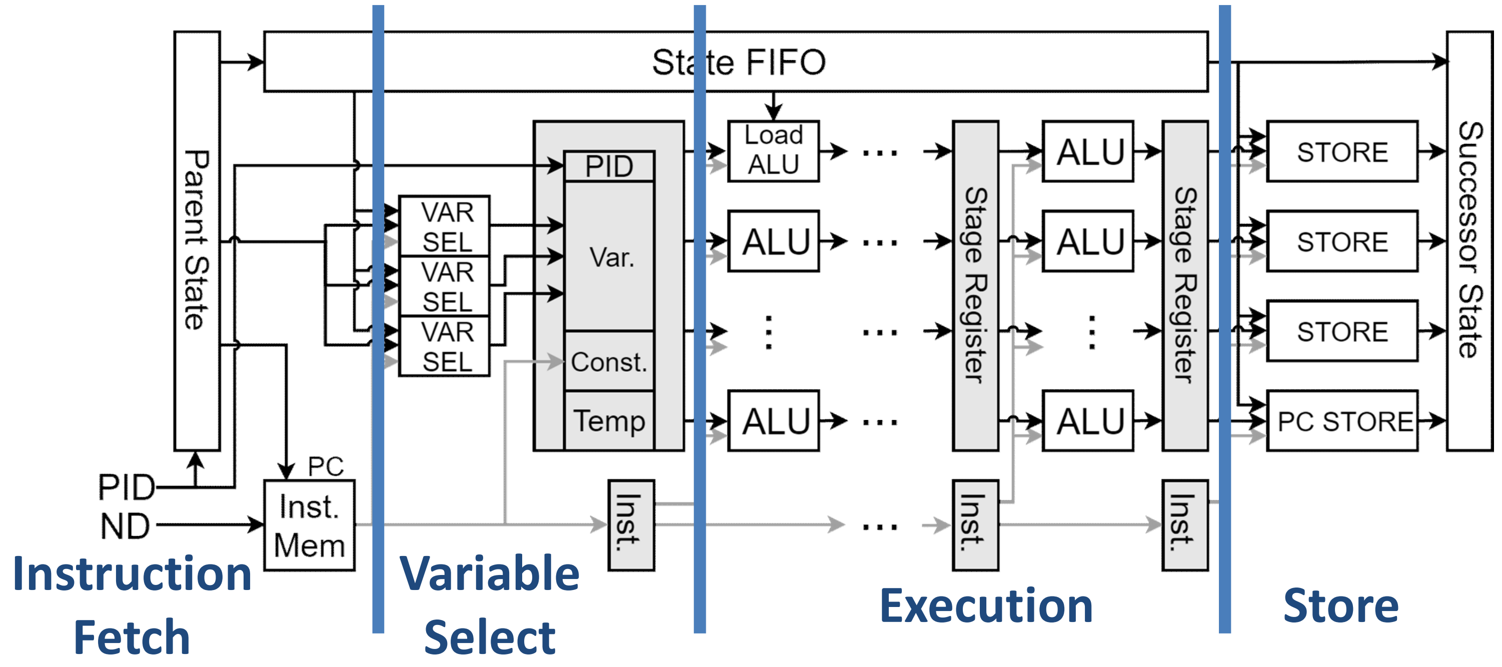
- Many independent model checker cores
- Control and violating state logging via AXI ports



Number of cores determines performance

Programmable Pipeline

- VLIW style pipeline
- 4 main stages for successor state generation
  - Instruction Fetch, Variable Select, Execution, Store

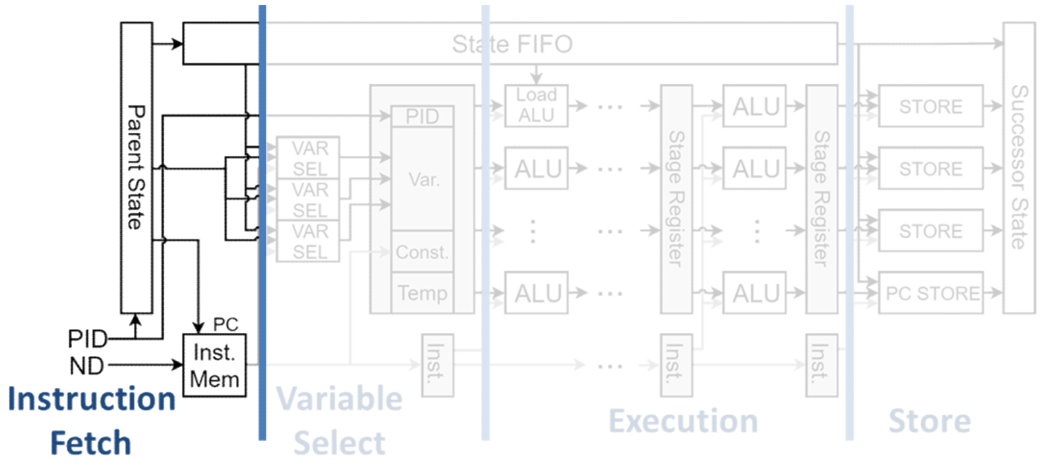


Stony Brook University

COMPAS

Instruction Fetch

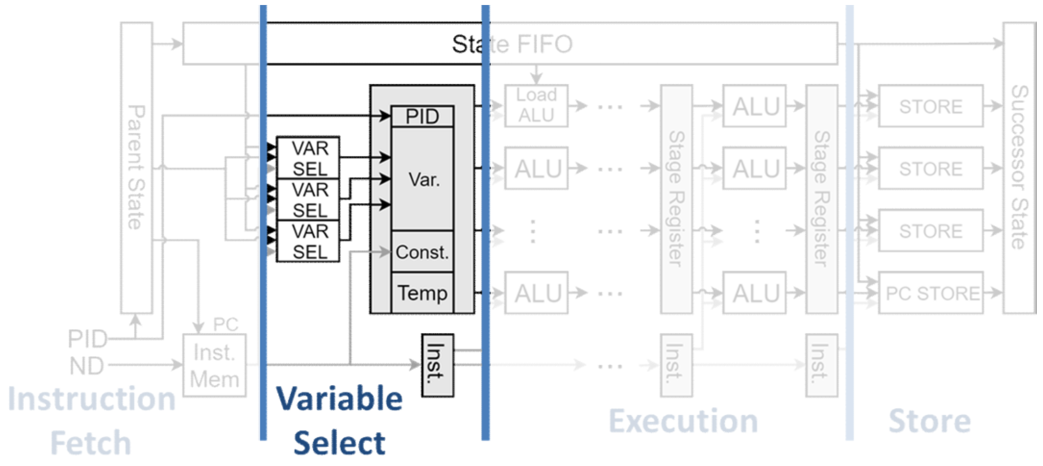
- Instruction contains control signals for following stages
  - Including constants for value calculation
- Instructions stored in BRAM
  - Guaranteed latency and one instruction per cycle
  - Independent access for model checker cores



Instructions increase per-core BRAM usage

Variable Select

- Load variables and constants required for calculation
  - Variables from the parent state vector
  - Constants from instruction
- Each variable select unit loads one variable
  - Number of select units dependents on models

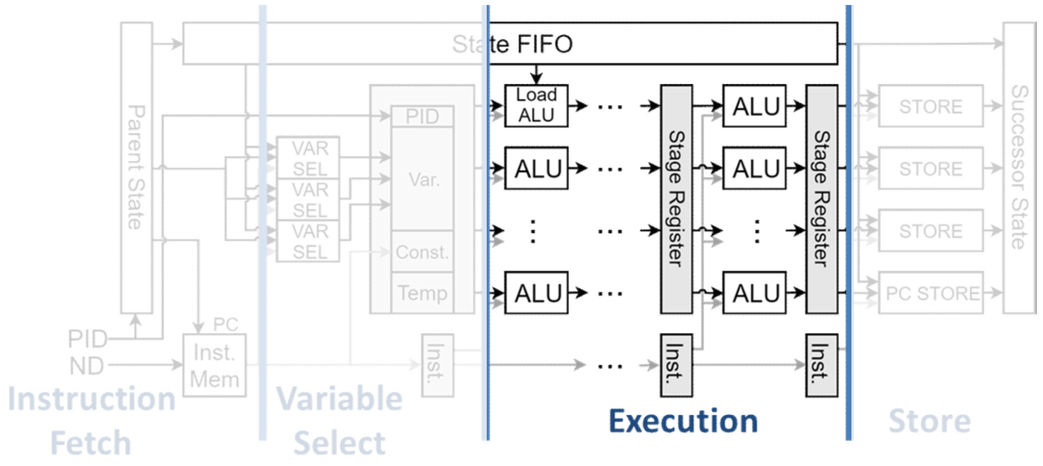


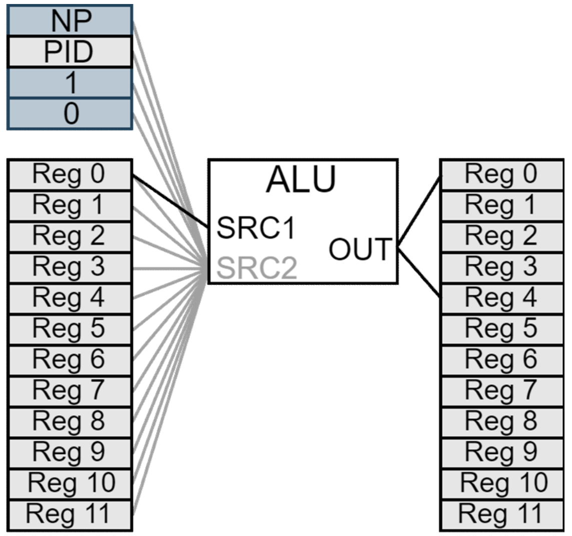
Stony Brook University

COMPAS

Execution

- A grid of ALUs to calculate:
  - Condition value
  - New variable values to be updated
- Limit ALU connection to reduce instruction length
  - … hence reduce BRAM usage





Stony Brook University

COMPAS

Execution

- Two types of ALUs:
  - Normal ALU for doing calculation
  - Load ALU for loading values from the state vector
    - Indexed array access
- Limit num of load ALU to reduce connection

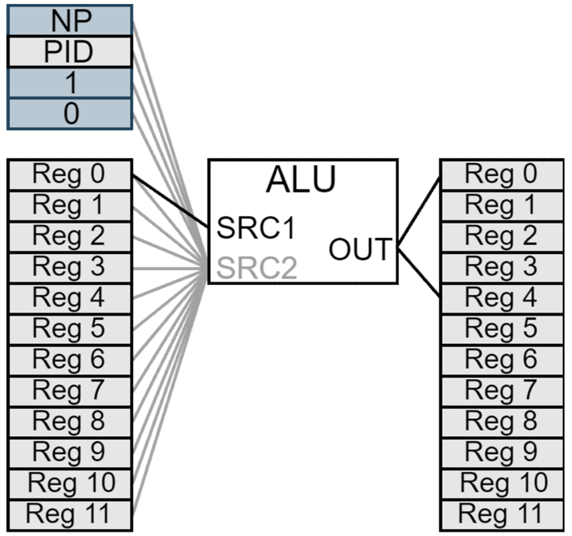
Normal ALU control bits

SRC2	OP	OUT

Load ALU control bits

BASE_SEL	PID	OP	OUT

PID=0: use SRC1 as index, PID=1: use PID as index



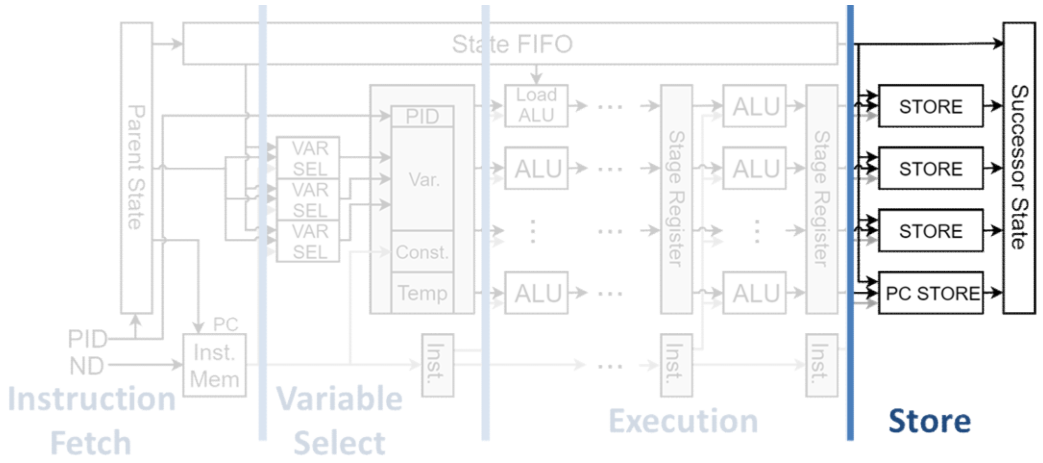
ALU designed to minimize connection and BRAM usage

Stony Brook University

COMPAS

Store

- Update variables inside the parent state vector
  - Based on condition calculated in the execute stage
- Each variable store unit updates variable
  - Number of store unites depends on models
- One PC store units dedicated for updating PC



Pipeline Parameters

- Stage requirements vary for different models
  - Variable select: number variables and constants
  - Execute: width and depth for the ALU grid
  - Store: number of variable that needs updating
- Affects the length of the instructions
  - … which affects BRAM usage per model checker core
  - … which affects number of cores can fit into an FPGA
  - … which affects performance

Overhead of programmability

Longer instruction → fewer cores → lower performance

Outline

Stony Brook University

COMPAS

Evaluation

- Programmable model checker on FPGAs
  - Programmable pipeline for successor generator
  - With overhead of programmability (fewer cores)
- Baseline model checker on FPGAs: FPGASwarm [Cho'18]
  - HLS based successor generator
  - No overhead of programmability (max num of cores)
- Common configurations for both model checker cores
  - Same frequency
  - Same per-core throughput (one-state-per-cycle)
  - Same queue size and visited state checker

Performance only depends on num of cores

Benchmarks

- 6 models from the BEEM database
  - Publicly available benchmark model set for model checkers

Benchmark	State Vec. (bytes)	Var. Sel. Units	ALUs Grid	Store Units	Inst. Size (bits)
Anderson.8	24	2	2x3	3	131
Bakery.8	28	2	2x5	3	167
Lamport.8	20	2	2x3	2	114
Leader_Filters.7	32	1	2x4	1	107
Mcs.6	24	2	1x1	2	64
Peterson.7	28	3	2x4	2	129

Stony Brook University

COMPAS

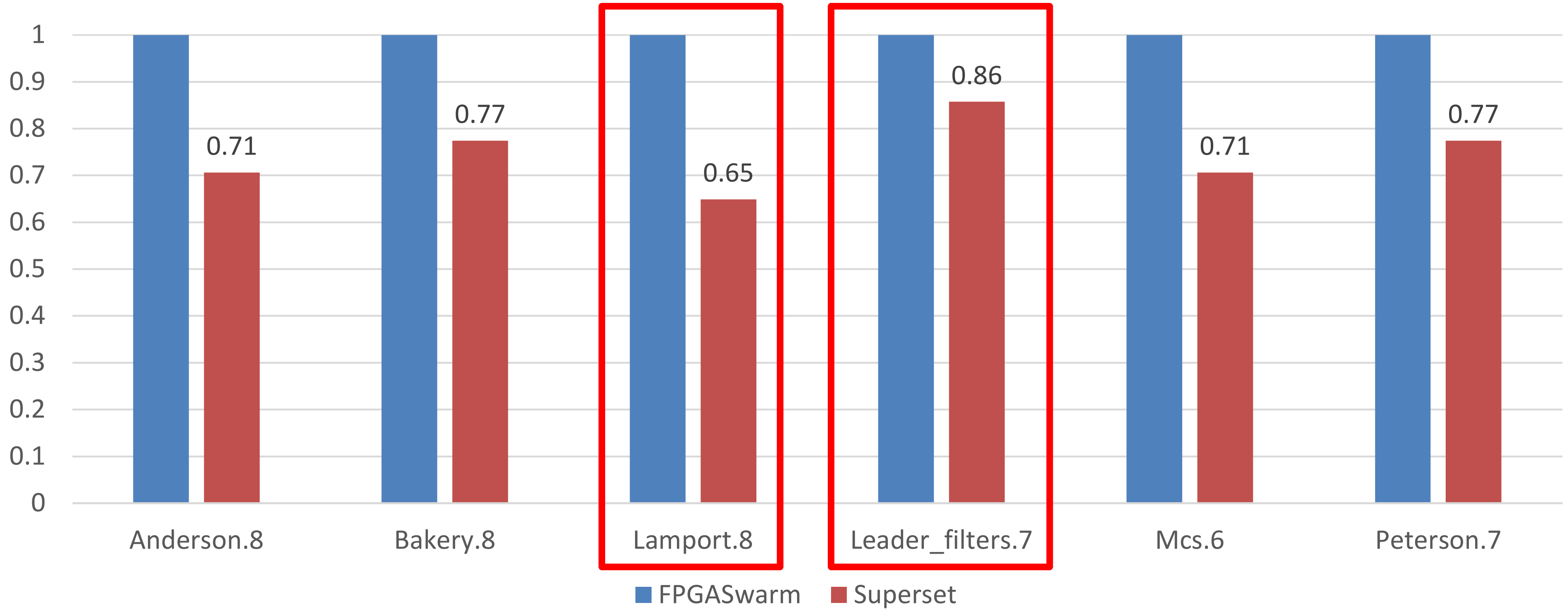
Results: Superset Checker for All

- One programmable checker for all benchmarks
  - Use the maximum parameter values
  - Load the model checker once for all benchmarks

Benchmark	State Vec. (bytes)	Var. Sel. Units	ALUs Grid	Store Units	Inst. Size (bits)
Anderson.8	24	2	2x3	3	131
Bakery.8	28	2	2x5	3	167
Lamport.8	20	2	2x3	2	114
Leader_Filters.7	32	1	2x4	1	107
Mcs.6	24	2	1x1	2	64
Peterson.7	28	3	2x4	2	129
Superset	**32**	**3**	**2x5**	**3**	**172**

Results: Superset Checker for All

- Maintain at least 60% run time performance
  - Still significant faster than software model checkers
- Waste BRAM for models with short state vectors



Unnecessary BRAM usage hurts performance

Optimization: Best-Fit Checkers

- Superset checker wastes BRAM from some models
- Solution: Pre-generate a model checker library

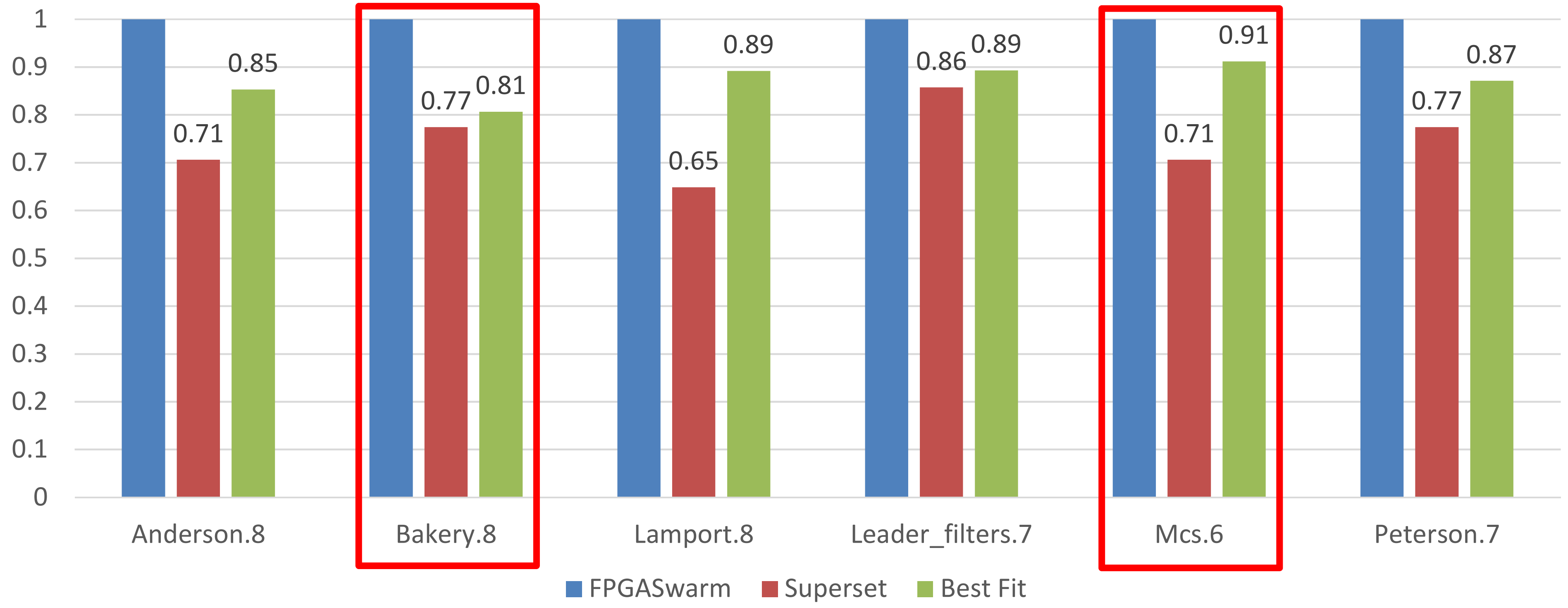
① Sweep parameters to pre-generate model checkers
  - State vector size, number of sub blocks in each stage
  - Does not affect preparation time or run time

② When given a model, analyze its parameters

③ Load the best-fit model checker to FPGA
  - With the closest parameters that can check that model

Optimization: Best-Fit Checkers

- Regain performance using best-fit checkers
  - Performance only affected by overhead of programmability



Recover 80% - 90% performance of prior work

Conclusions

- Model checker on FPGAs shows significant speedup
  - Orders-of-magnitude speedup in run time from prior work
- But the FPGA preparation time voids the speedup
  - Synthesis and P&R required for every new/modified models
- Programmable pipeline eliminates preparation time
  - Avoid synthesis and P&R
  - Pre-compiled best-fit bitstreams to minimize overhead
  - Maintain 80% - 90% of the run time performance

Stony Brook University

COMPAS

Backups

Stony Brook University

COMPAS

Backups

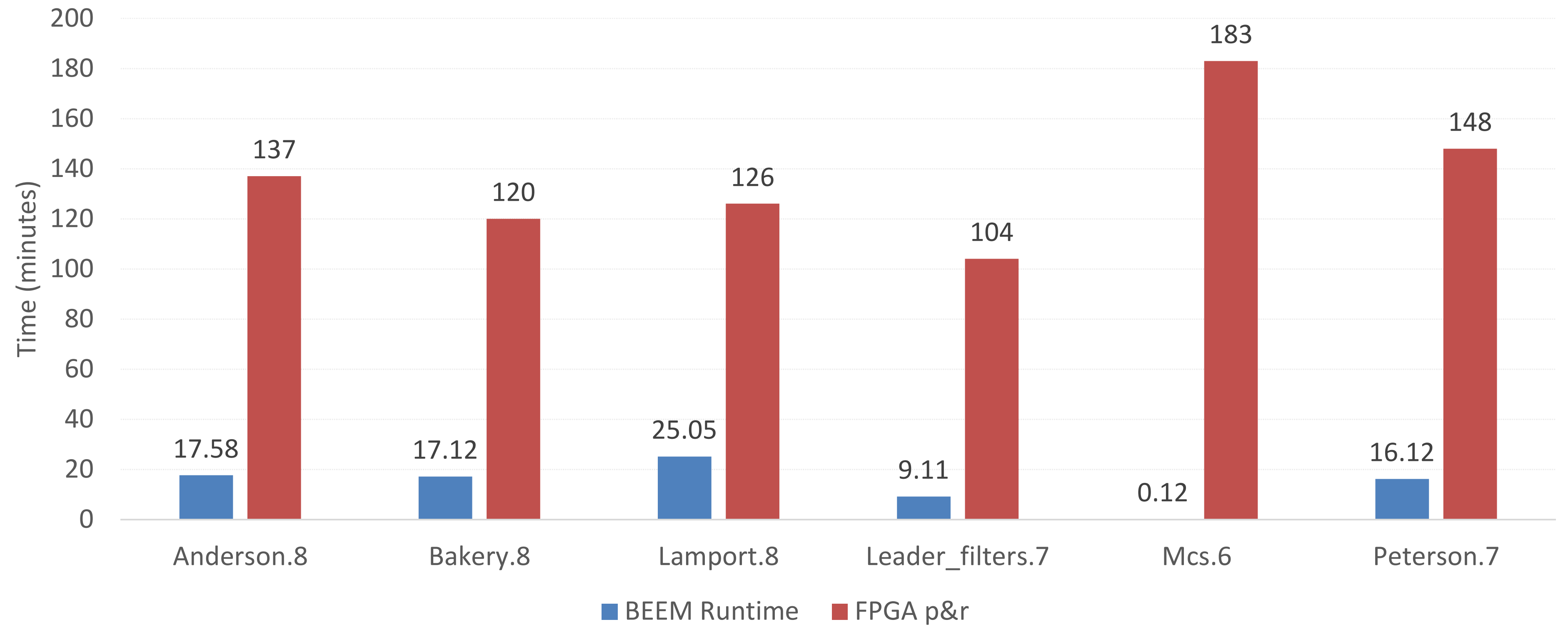
Backups

Stony Brook University

COMPAS

Model Checking Time

- FPGA preparation time is significantly longer than model checking runtime by software



Background

- Promela
  - ND: Non-determinism factor
  - PC: Current state
  - PID: Process ID

```
byte balance=1;
active [2] proctype customer() {
  byte cash=0;
  S: if :: goto W;
        :: goto end;
  fi;
  W: if :: d_step { balance=balance-1;
                  cash=cash+1; };
           goto end;
  fi
  end:
}
```

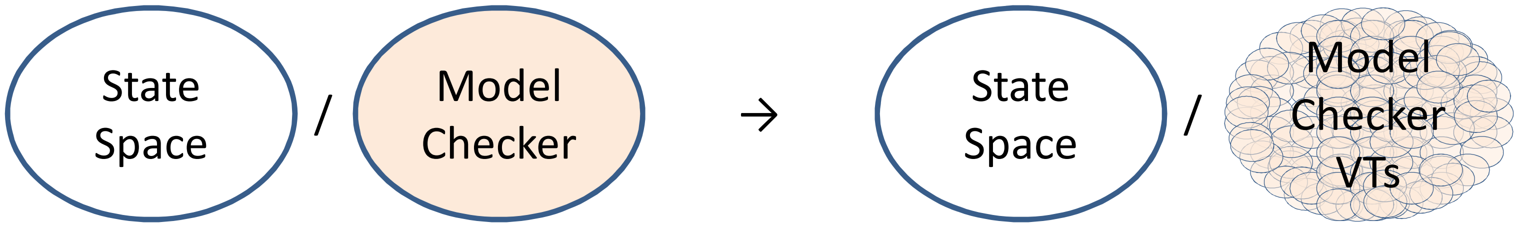
Instruction Fetch

- address <= {PC, ND};
- Instruction format

Selection			Execute 0			Execute 1			Execute 2			Store		
Unit 0	…	Unit m	ALU 0	…	ALU n	ALU 0	…	ALU n	ALU 0	…	ALU n	Unit 0	…	Unit o

(Software) Swarm Verification [Holzmann'08]

- Expose parallelism in model checkers
  - Replace one large model checker with many small ones
  - Each "verification task" (VT) explores part of state space
    - VTs will overlap in exploration
    - … but combination will statistically cover the space
- Advantages:
  - Massive, completely independent parallelism
  - Memory usage per model checker: GBs → MBs

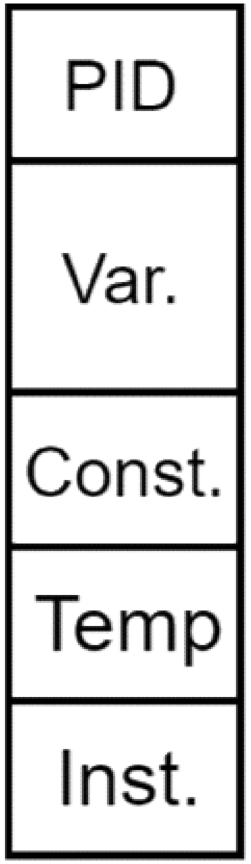


Stony Brook University

COMPAS

Pipeline Stage Registers

- PID
- M selected values
- N immediate values (Constants)
- Several temporaries
- Instruction


PID
Var.
Const.
Temp
Inst.