An End-to-End Approach for the Automatic Derivation of Application-Aware Error Detectors

Galen Lyle

Shelley Chen

Karthik Pattabiraman

Zbigniew Kalbarczyk

Ravishankar Iyer

Motivation : Error Detection

- **Existing techniques for error detection (e.g. duplication)**
  - Incur substantial overheads in space or time (60 to 100 %)
  - Detect errors that do not impact application (benign errors)
- **Goal: Detect errors that matter to the application before propagation (at low cost)**
- **Our Approach: Application-aware Detection**
  - Error detectors based on properties of the application
  - Automated process for deriving and implementing error detectors

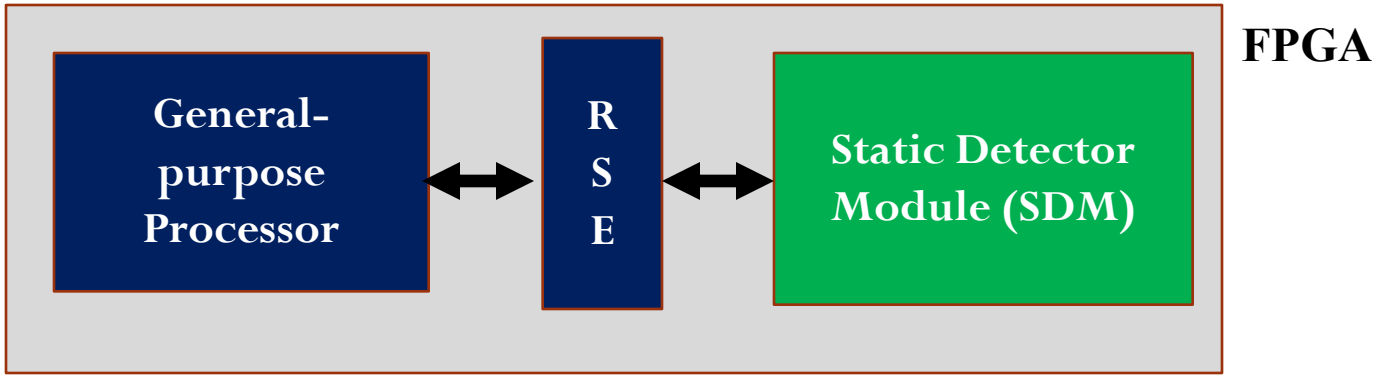
Earlier Work: Critical Variable Recomputation (CVR)

- **Insertion of runtime checks in the program**
  - Automatic analysis through new compiler pass
  - Checks synthesized as custom software libraries
- **Coverage evaluation using fault-injections**
  - Detection coverage = 77 % (for failure-causing errors)
  - Less than 3 % benign error detections
- **But, performance overheads ….**
  - Up to 141 % on Pentium-4 processor (average = 33 %)
  - Up to 555 % on Leon-3 (due to limited parallelism)

This Paper: H/W Implementation

- **Hardware implementation to reduce overheads**
  - Monitoring/checking in parallel with application

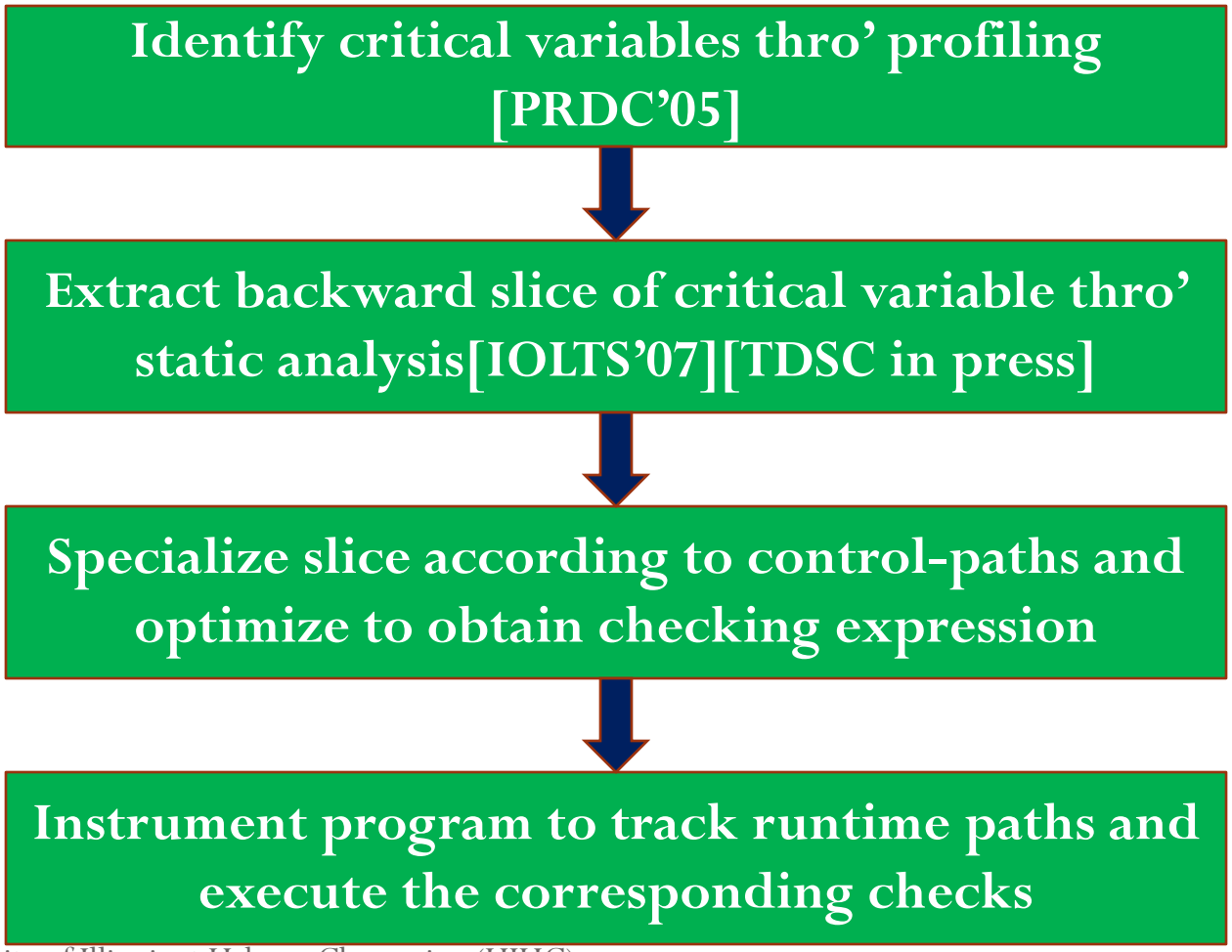
- **Implemented using Reconfigurable Hardware (FPGAs)**
  - Application executes on general-purpose processor
  - Checks execute on separate module - **Static Detector Module**
  - Module interfaces with processor through generic interface (RSE)



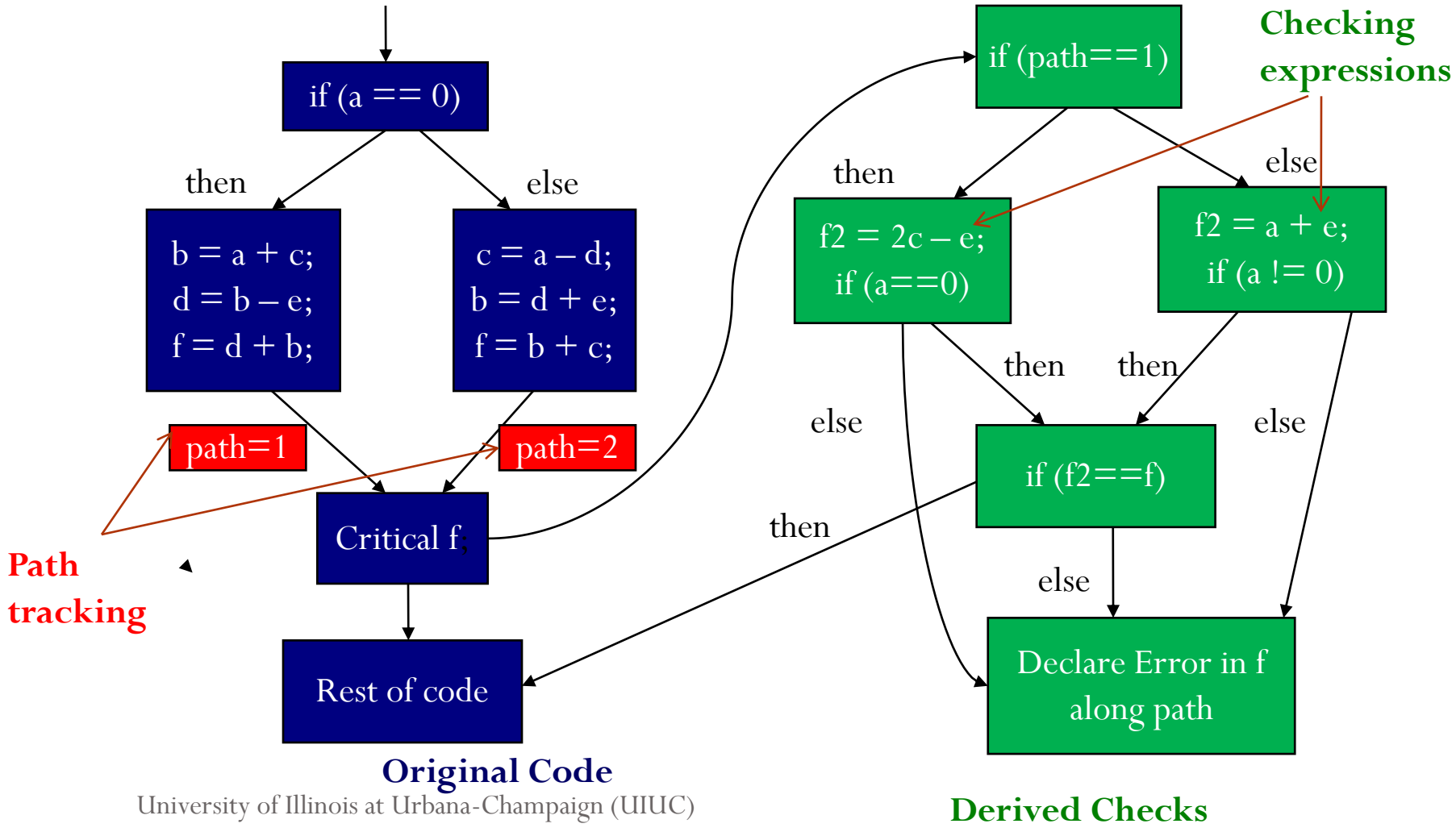
Paper Outline

- Motivation and Approach
- **CVR Technique (Recap)**
- H/W Implementation
- Experimental Results
- Conclusions and Future Work

CVR Technique: Overall Algorithm



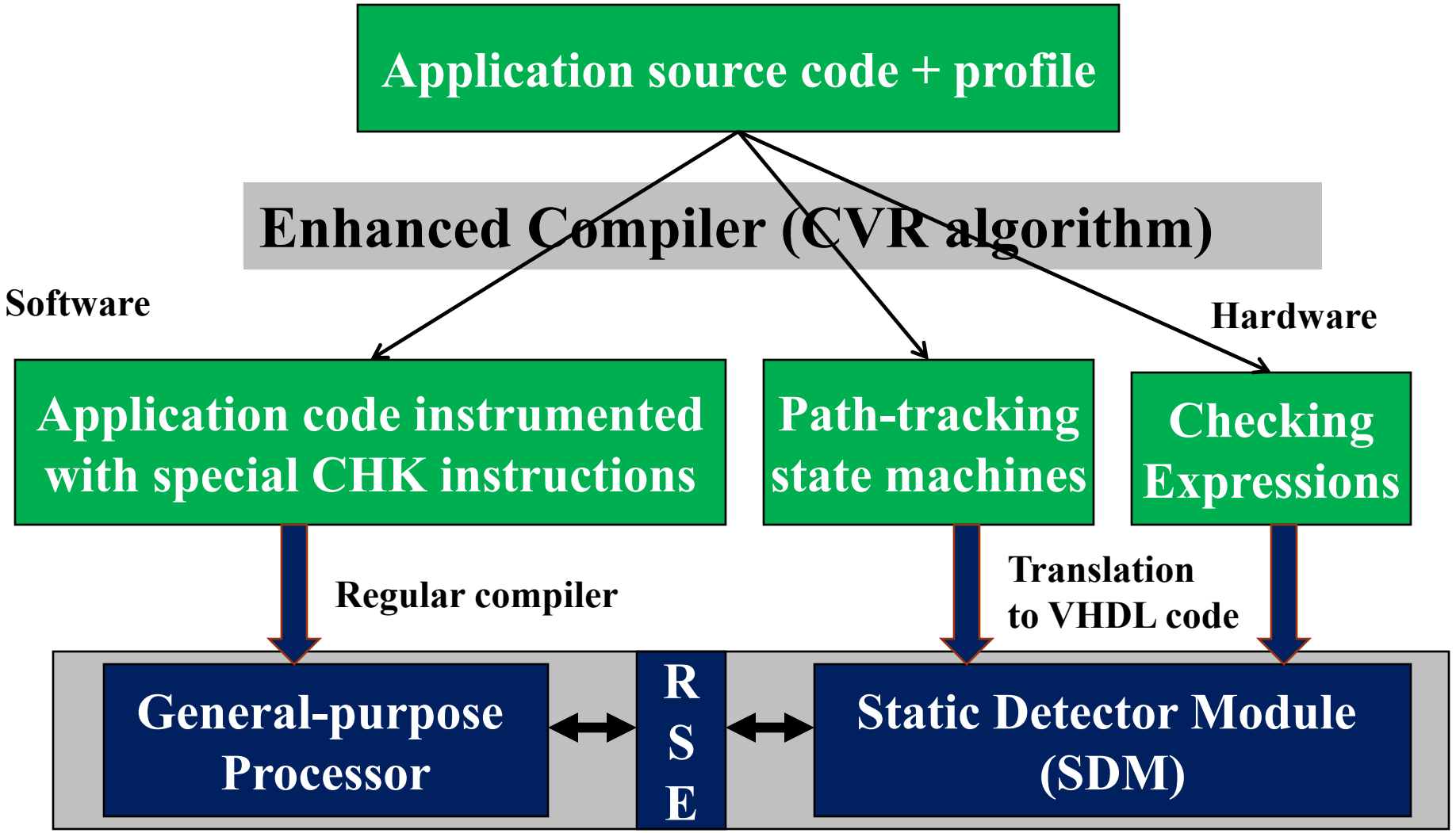
CVR Technique: Example



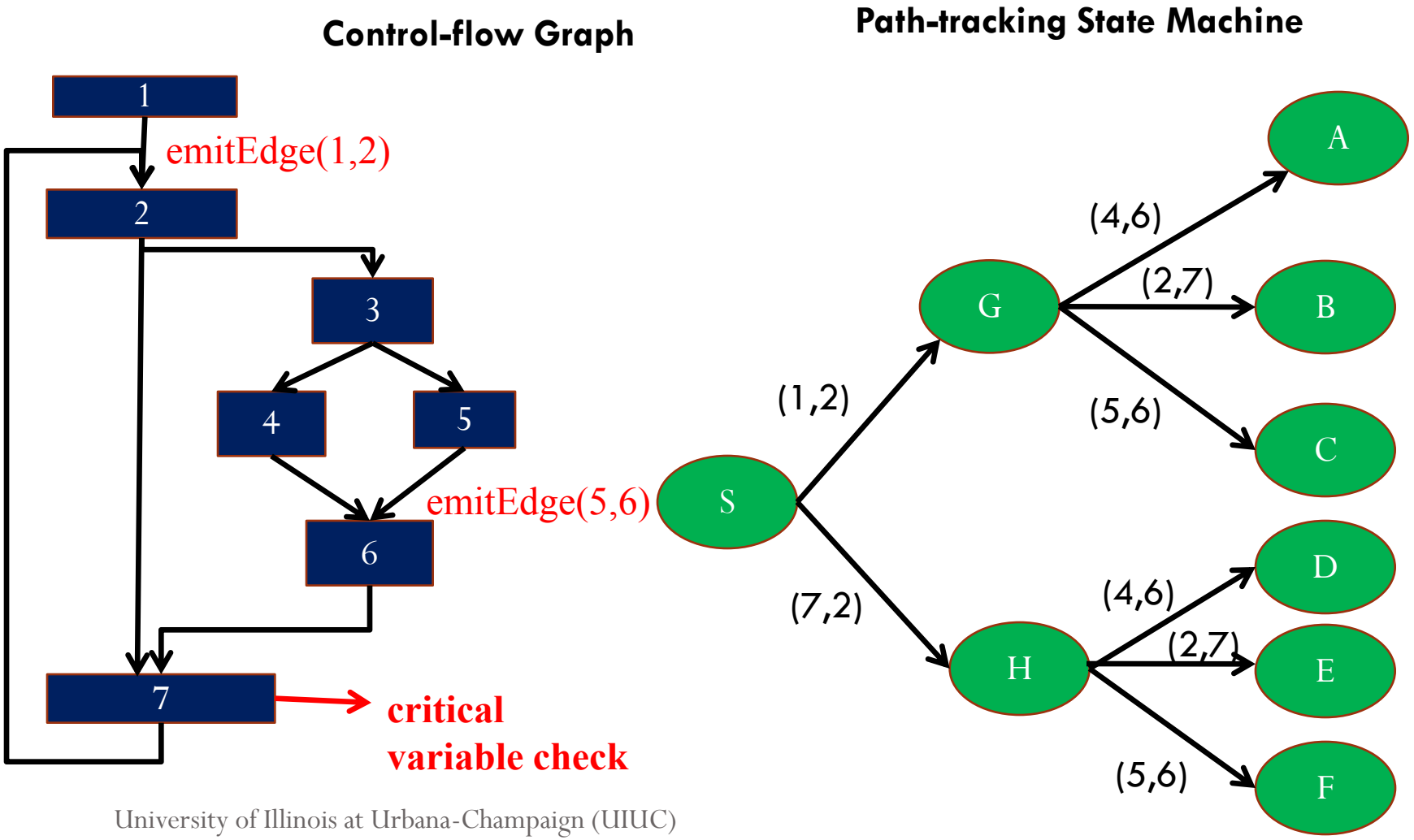
Paper Outline

- Motivation and Approach
- CVR Technique (Recap)
- **H/W Implementation**
- Experimental Results
- Conclusions and Future Work

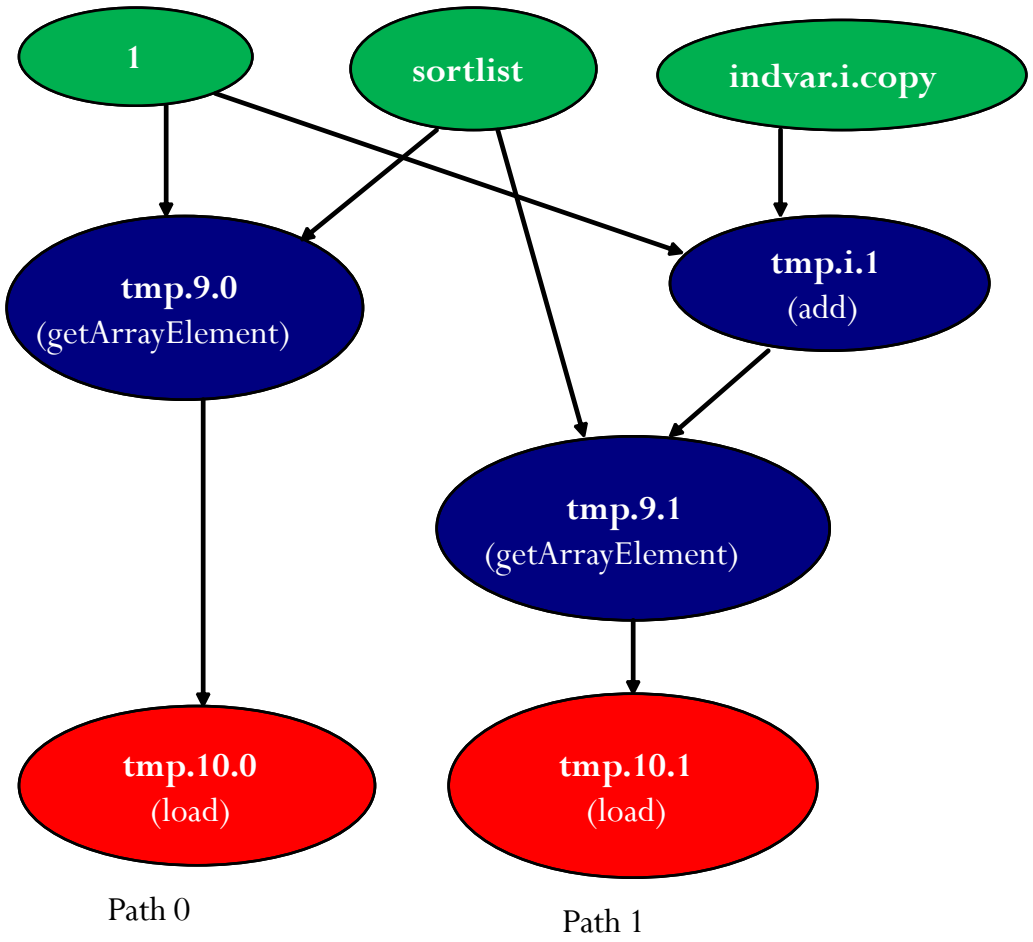
H/W Implementation: Design Flow



H/W Implementation: Path Tracking



H/W Implementation : Checking Expression



Inputs: Passed in sharedARGQ

sortlist

indvar.i.copy

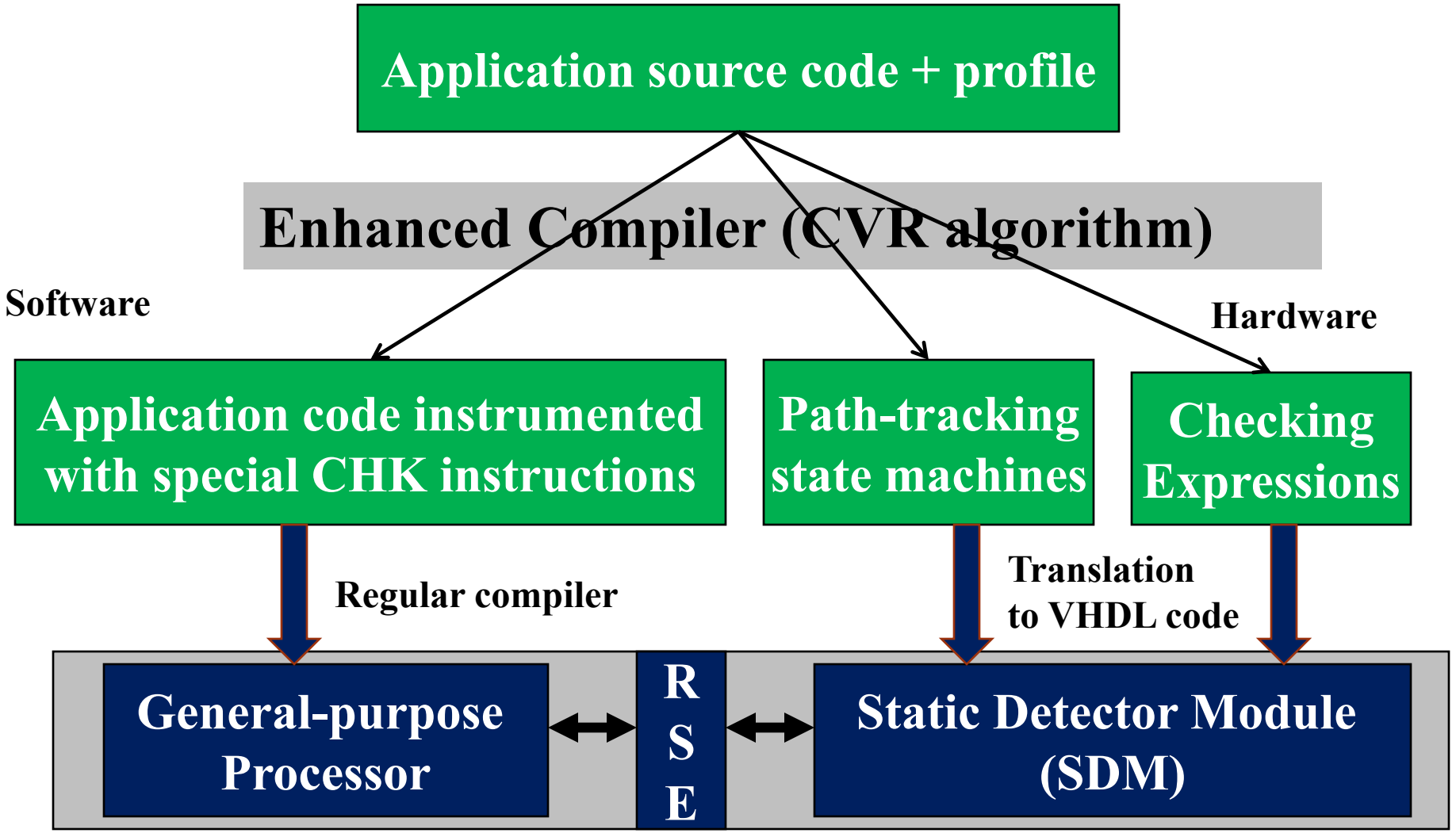
Instructions : Converted to DAG

t0 = 1
t1 = sortList
t2 = getArrayElement t0,t1
t3 = load (*t2)
t4 = indvar.i.copy
t5 = t0 + t4
t6 = getArrayElement t5, t1
t7 = load (*t6)

Path 0 : Final = t3

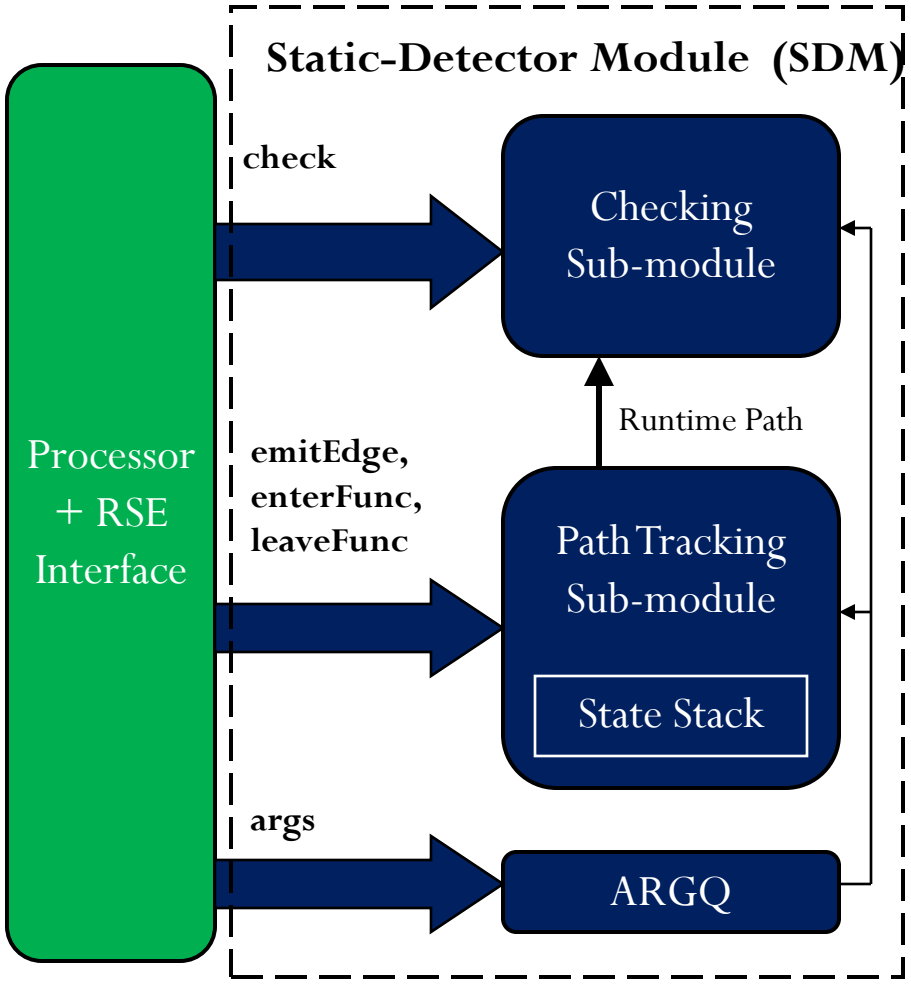
Path 1 : Final = t7

H/W Implementation: Design Flow

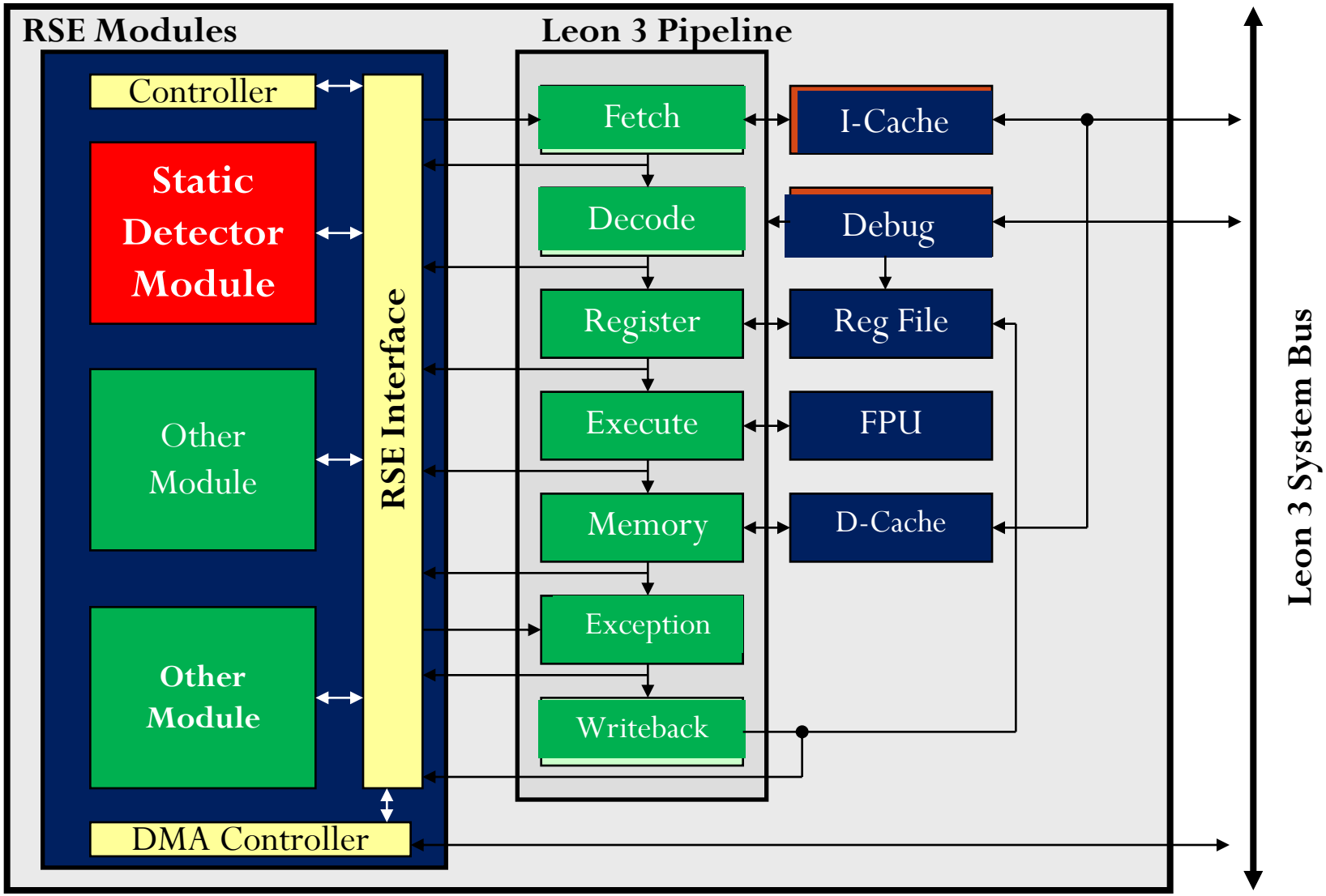


H/W Implementation: SDM

- **Static Detector Module (SDM)**
  - Path-tracking
  - Check execution
- **Path-tracking sub-module**
  - Monitors *emitEdge* operations
  - State Stack for function calls
- **Checking sub-module**
  - Parameter passing: ARGQ
  - Direct access to memory



H/W Implementation: RSE



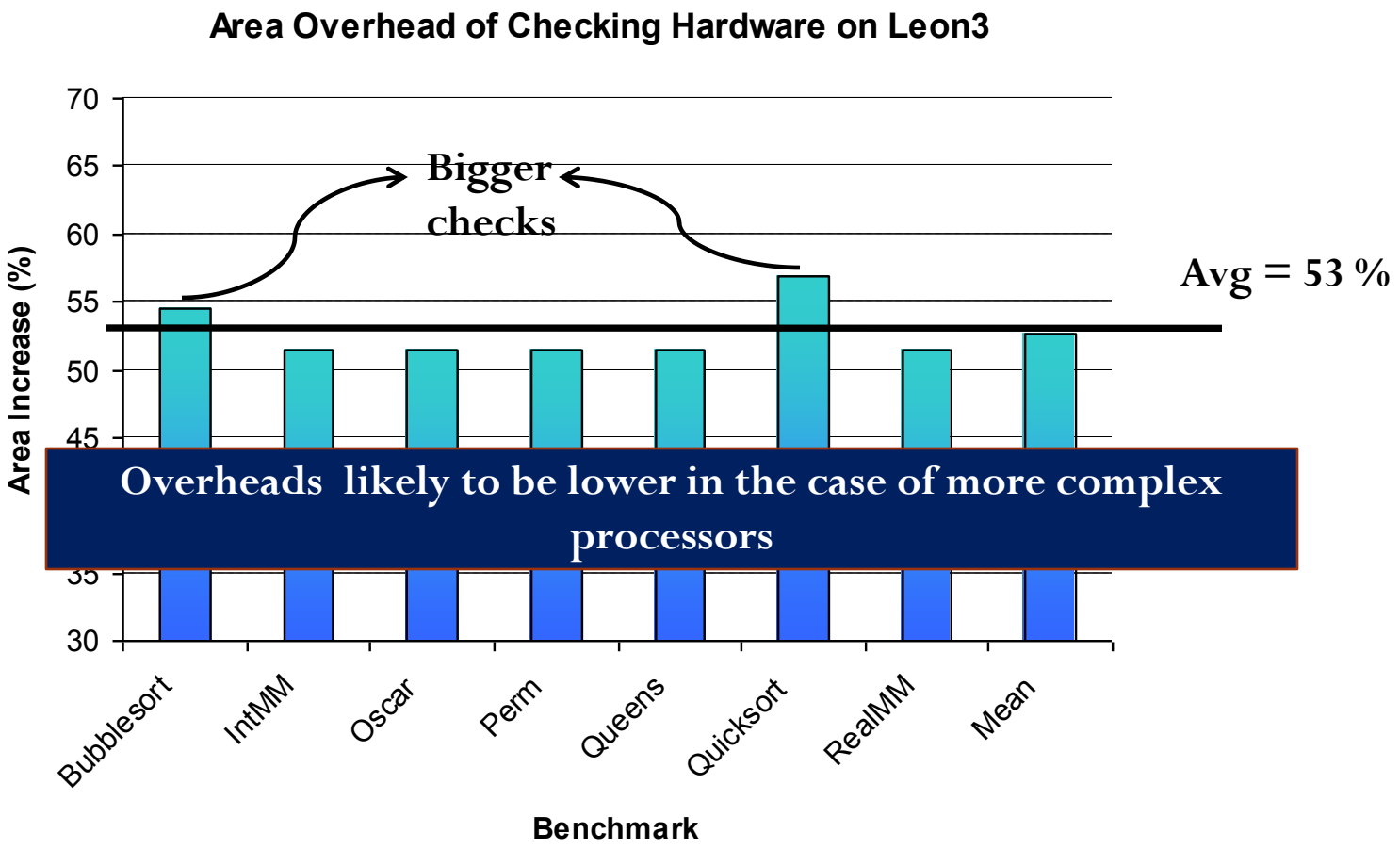
Paper Outline

- Motivation and Approach
- CVR Technique (Recap)
- H/W Implementation
- **Experimental Results**
- Conclusions and Future Work

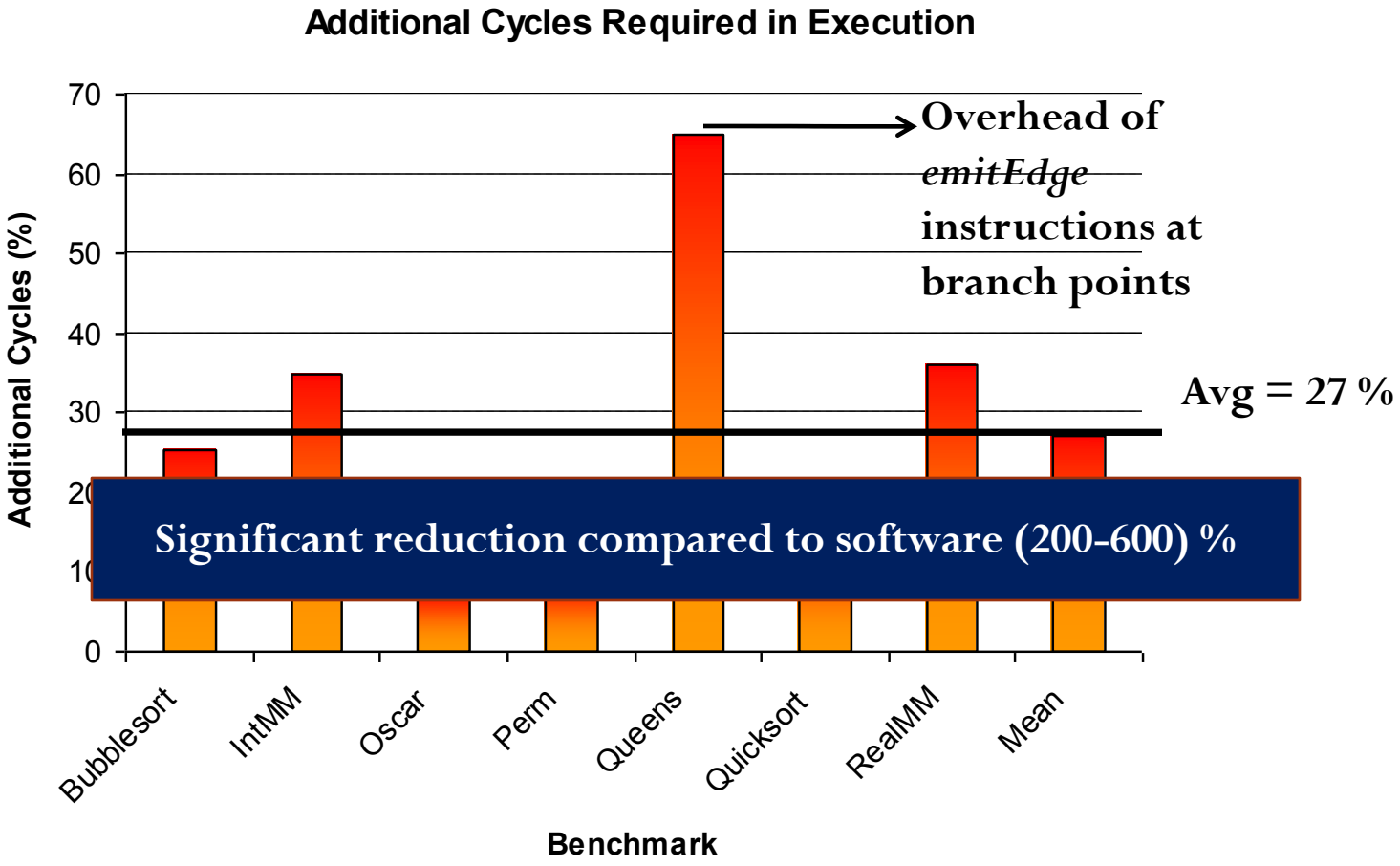
Experimental Setup

- **Benchmarks: Stanford Programs**
  - 100-500 lines of C code
- **VHDL generated according to check and path definitions obtained during compilation**
  - VHDL simulation in ModelSim 6.3
  - Hardware synthesis for Xilinx Virtex-2 Pro FPGA
- **Measurement of area and performance overheads using direct execution on FPGA hardware**
  - Includes overheads of RSE interface (but no other modules)

Results (Area Overheads)



Results (Performance Overheads)



Paper Outline

- Motivation and Approach
- CVR Technique (Recap)
- H/W Implementation
- Experimental Results
- **Conclusions and Future Work**

Conclusions

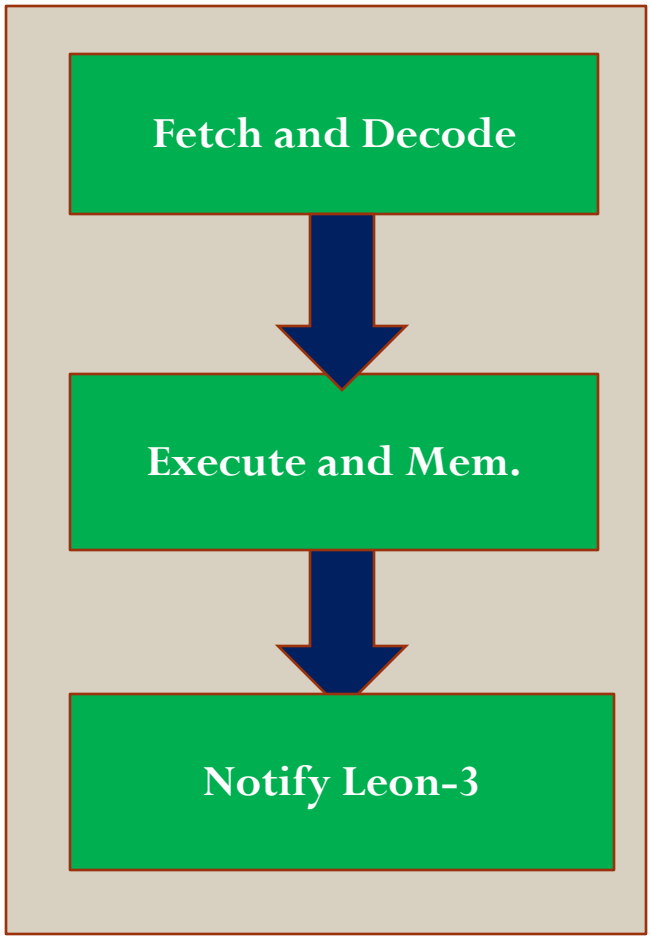
- **Application-aware detection is a low-overhead approach for providing high detection coverage**
  - Compiler support to derive detectors (checks)
  - Reconfigurable hardware support for executing checks

- **H/W Implementation as RSE module**
  - Synthesized and implemented on FPGA
  - Area overheads bet. 50-55 % **(Avg = 53 %)**
  - Perf. Overheads bet. 15-65 % (**Avg = 27 %**)

Low performance overheads with SDM, but potentially unbounded area overheads

Ongoing work: Micro-controller

- **Our idea**
  - Use a small micro-controller in place of the SDM
- **Programmed at application load time**
  - No need for reconfiguration
- **Results**
  - Perf overhead = 30 % (Vs. 27 %)
  - Area overhead = 45 % (const)

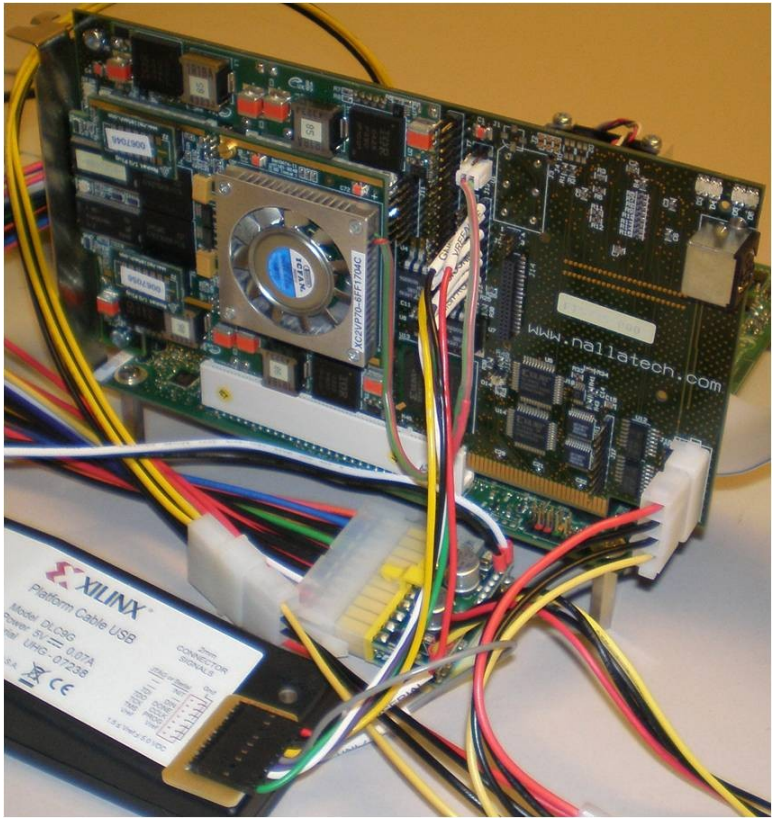


Micro-controller

Future Work

- **Feedback loop from hardware to compiler for estimating performance and area overhead apriori**
- **Eliminate *emitEdge* instructions by observing branches (reduction of performance overheads)**
- **Identify common state across checks and optimize these out (reduction of area overhead)**

Trusted ILLIAC Project (UIUC)



Custom hardware as extension cards

Trusted Illiac Node for prototyping and development

Projected Vision

•256 nodes

Thank You

Questions ?

Contact: Karthik.Pattabiraman@gmail.com