Verifying Timing-Centric Software Systems

Sanjit A. Seshia

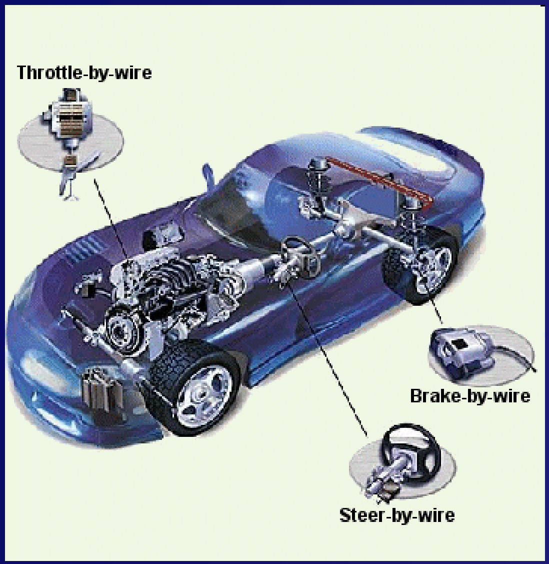
EECS Department

UC Berkeley

Students: **Jon Kotker**, **Dorsa Sadigh**,
Sagar Jain, Min Xu, Andrew Chan, Lisa Yan
Collaborators: A. Rakhlin
Funding Sources: NSF, MuSyC, Industry sponsors

May 2011

Timing Analysis is Central to Correctness of Cyber-Physical Systems



Does the brake-by-wire software always actuate the brakes within 1 ms?

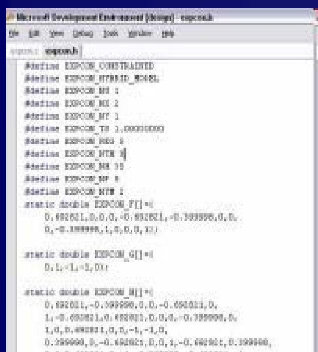
Can the pacemaker software trigger a pace more frequently than prescribed?

Several Timing Analysis Problems

- **Worst-case execution time (WCET) estimation**
- **Estimating distribution of execution times**
- **Threshold property: produce a test case that causes a program to violate its deadline**
- **Software-in-the-loop simulation: predict execution time of particular program path**

ALL involve predicting timing for some/all executions of a program!

What’s Hard about Timing Analysis



PROGRAM → Timing Analysis Tool → Timing estimate (e.g. worst case)

UNDECIDABLE PROBLEM!

However: Only need to consider terminating programs
- statically-known bounds on loop iterations, recursion depth

Example of Software Task

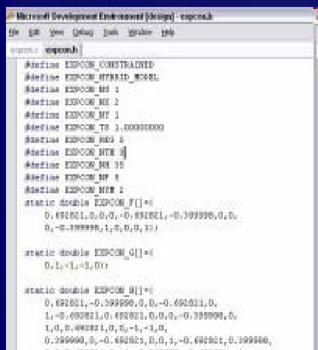
altitude_control_task() from implementation of software controller of "Paparazzi UAV"

```
main.c:
…
while(1) {
  …
  periodic_task(…);
  …
}
```

```
switch(…) {
  case 0: …
     altitude_control_task(…);
     …
}
```

```
void altitude_control_task(void) {
 if (pprz_mode == PPRZ_MODE_AUTO2
     || pprz_mode == PPRZ_MODE_HOME) {
  if (vertical_mode == VERTICAL_MODE_AUTO_ALT) {
  /* inlined below: function  altitude_pid_run(); */
    float err = estimator_z - desired_altitude;
    desired_climb = pre_climb + altitude_pgain * err;
    if (desired_climb < -CLIMB_MAX)
      desired_climb = -CLIMB_MAX;
    if (desired_climb > CLIMB_MAX)
     desired_climb = CLIMB_MAX;
}}}
```

What’s Hard about Timing Analysis



TERMINATING PROGRAM

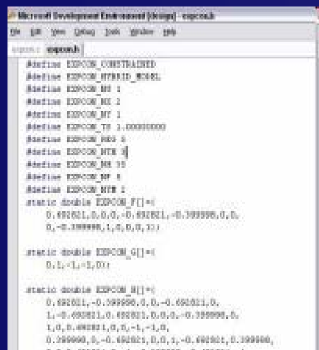
→ Timing Analysis Tool → Timing estimate (e.g. worst case)

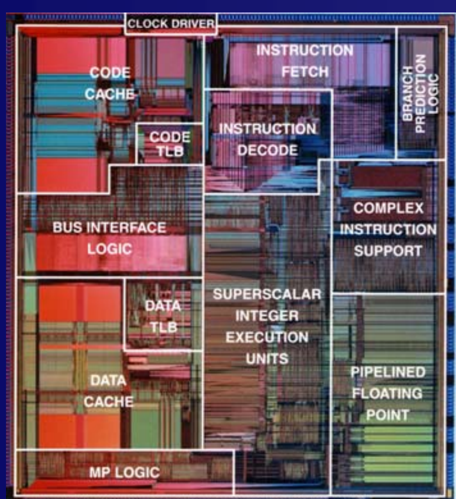
Program timing depends HEAVILY on the program’s environment (platform): processor, memory hierarchy, operating system, network, I/O devices, …

Outline

- Challenge: Platform (Environment) Modeling
- The GameTime Approach: Learning Program-Specific Environment Model
- Conclusion and Future Directions

Simplifying the Problem



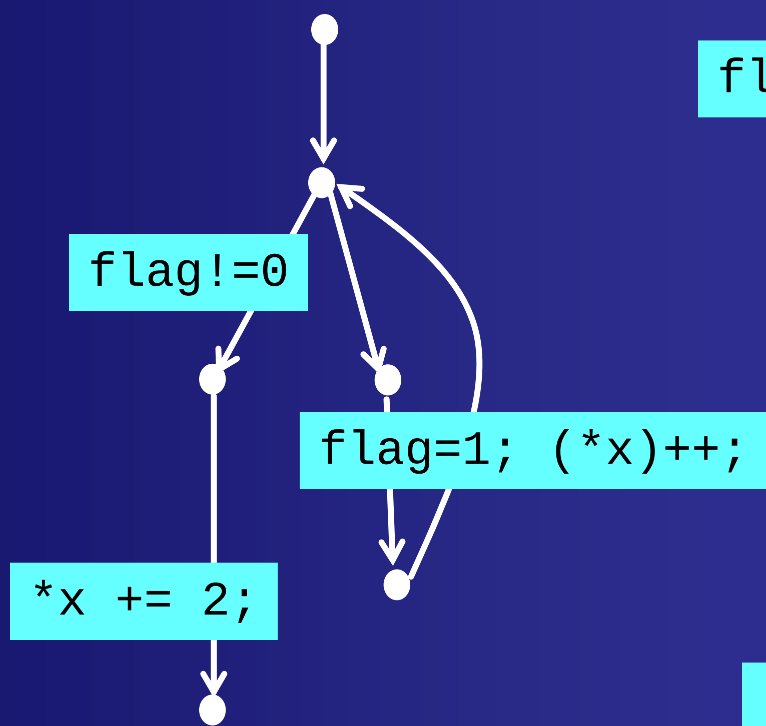


- **Program = Sequential, terminating program**
- **Runs uninterrupted**

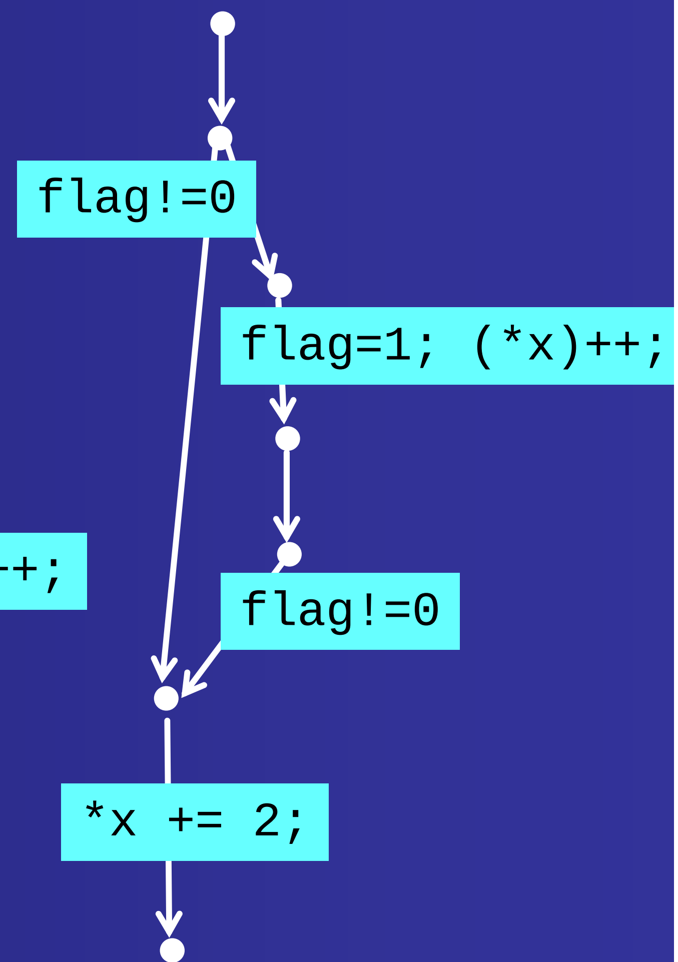
- **Environment = Single-core Processor + Instruction/Data Cache**

Simple Illustrative Program

```
while(!flag)
{
   flag = 1;
   (*x)++;
}
*x += 2;
```



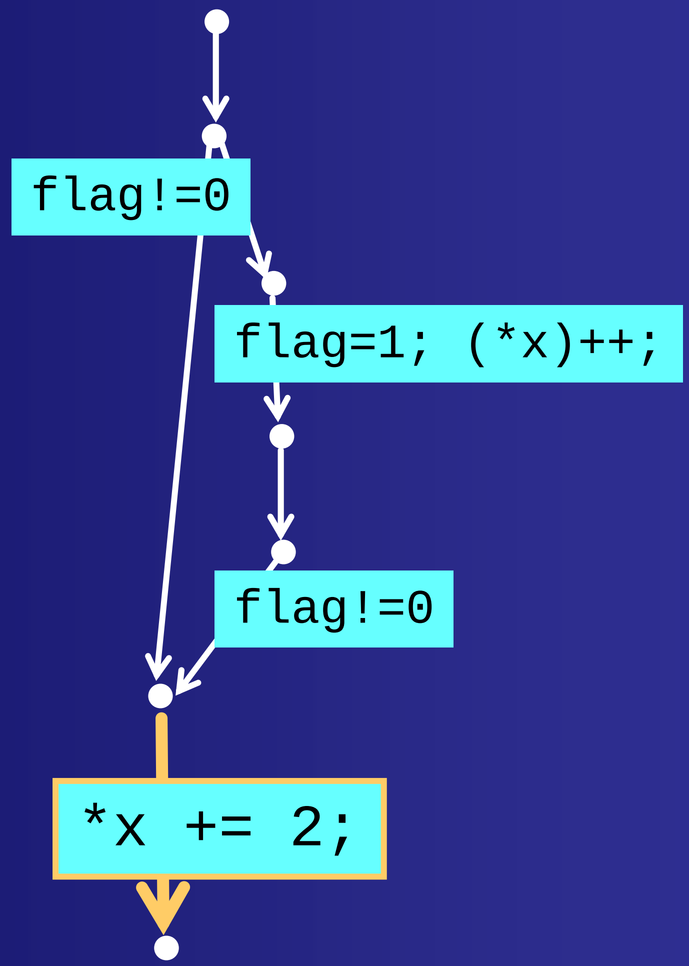
Control-flow graph
(loop bound = 1)



CFG unrolled to a DAG

Simple Illustrative Program

On a processor
with a data cache

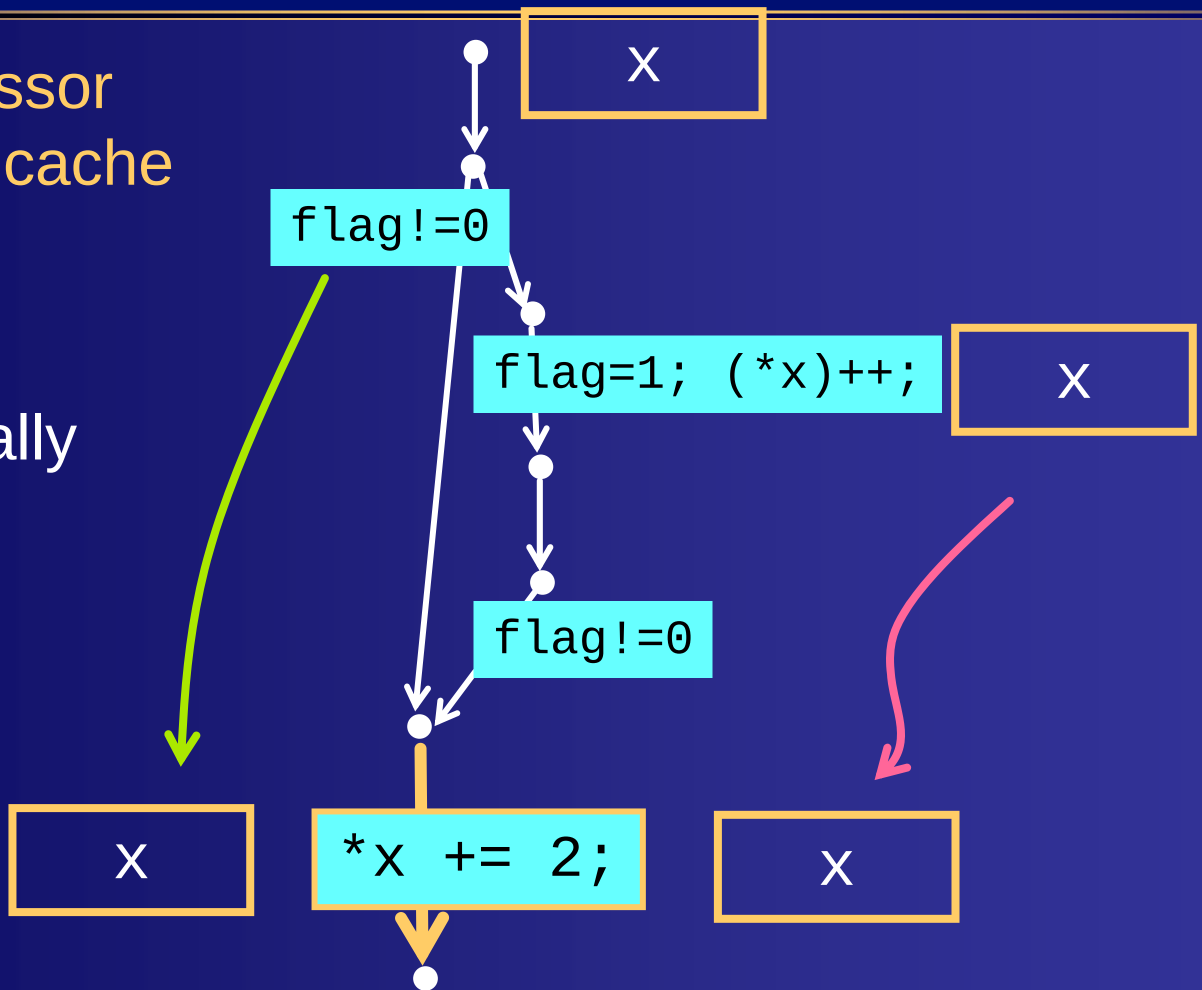


CFG unrolled
to a DAG

Simple Illustrative Program

On a processor
with a data cache

Case 1:
x is originally
in cache

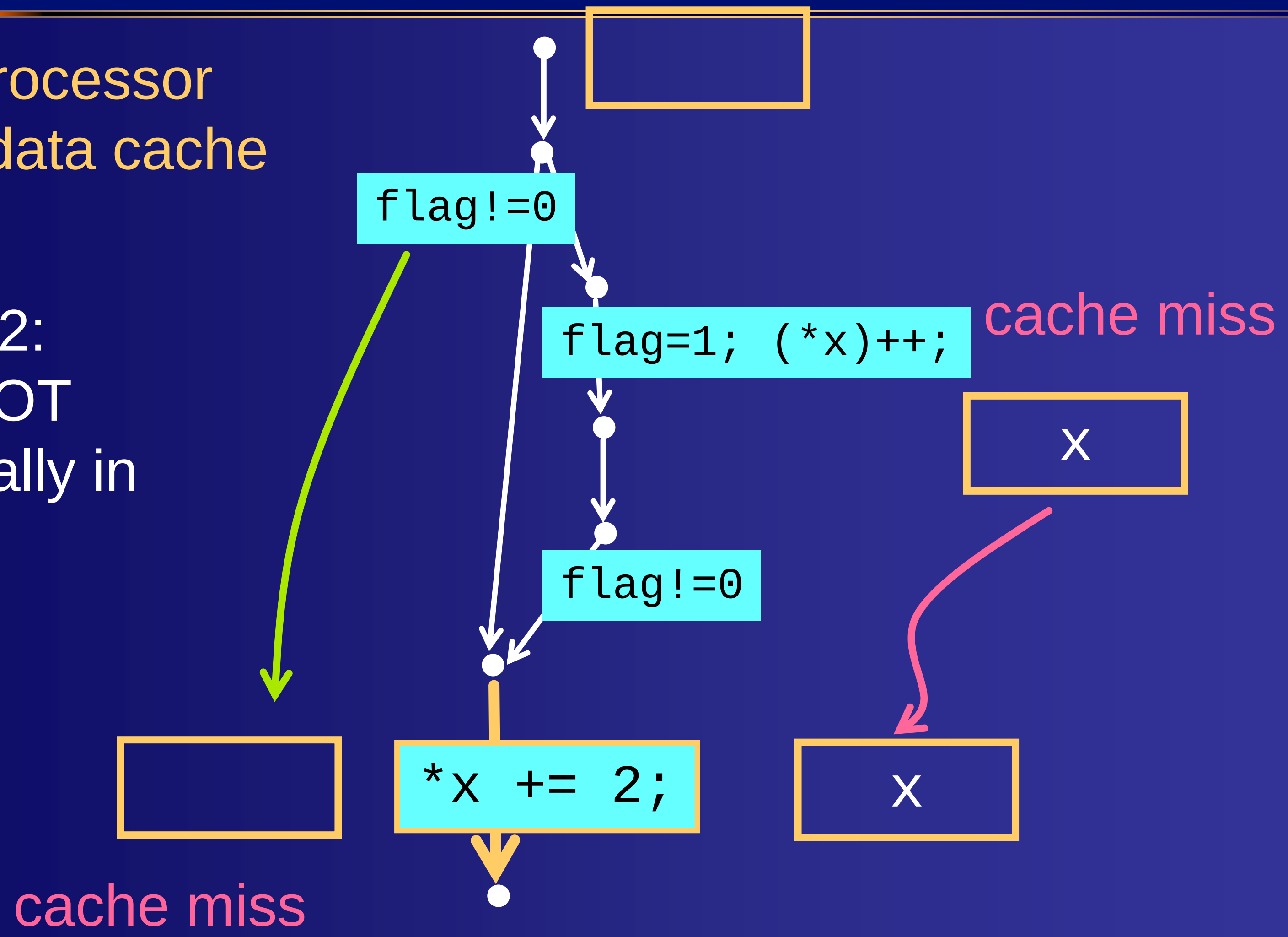


CFG unrolled
to a DAG

Simple Illustrative Program

On a processor
with a data cache

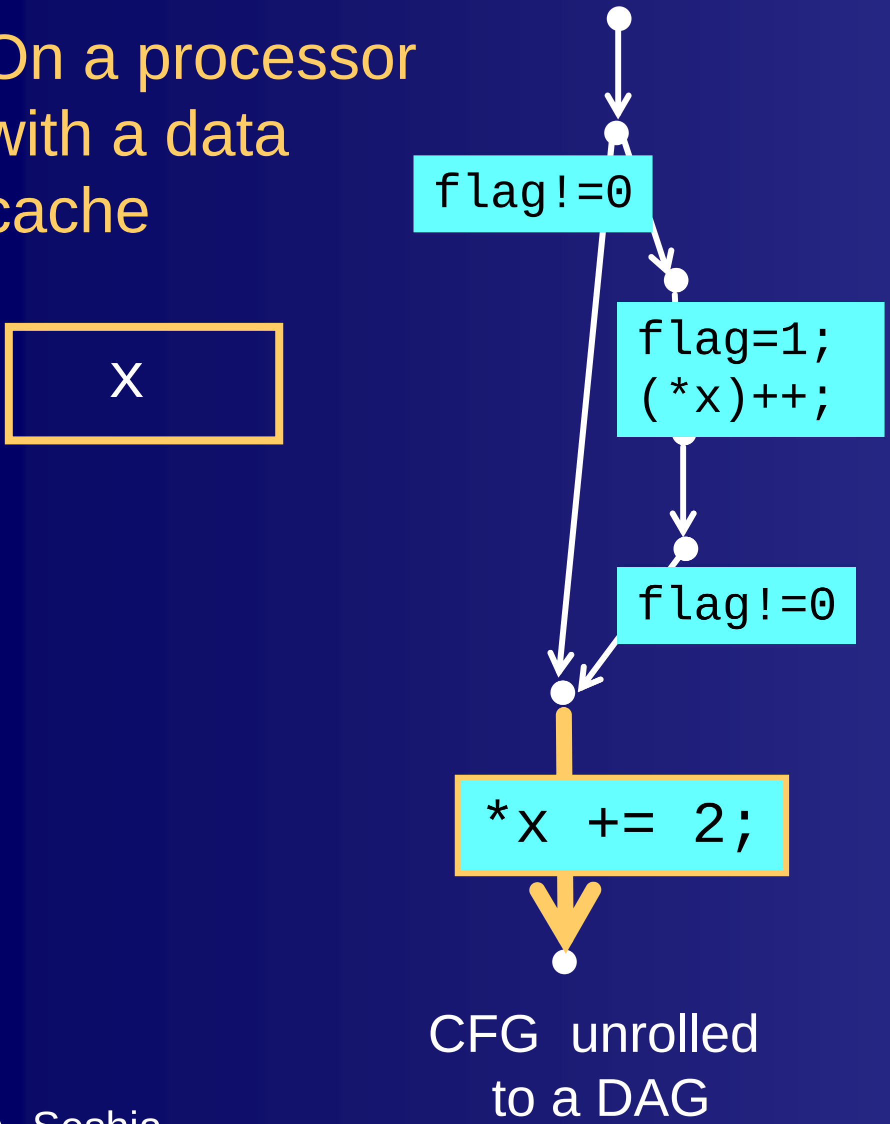
Case 2:
x is NOT
originally in
cache



CFG unrolled
to a DAG

Challenge of Timing Analysis

On a processor with a data cache



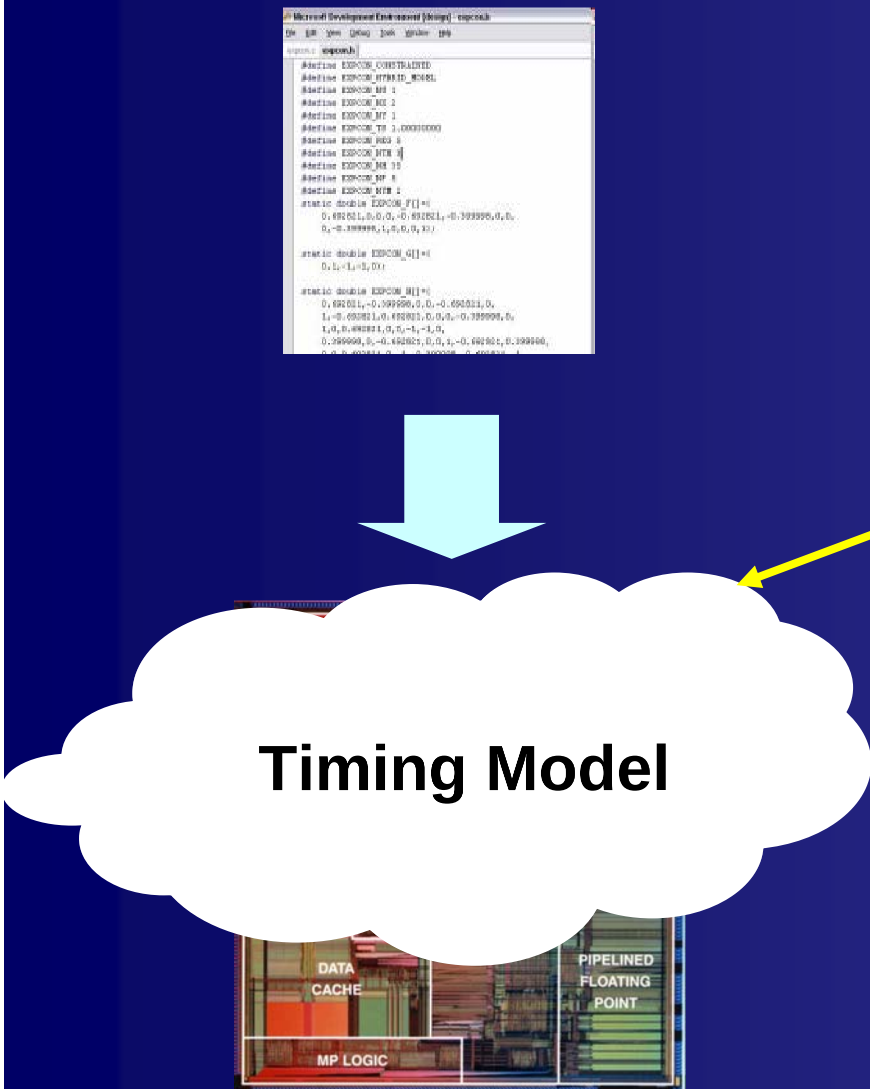
Timing of an edge (basic block) depends on:

- **Path** it lies on
- Initial **platform state**

Challenges:

- Exponential number of paths and platform states!
- Lack of visibility into platform state

Current State-of-the-art for Timing Analysis



- **Program = Sequential, terminating program**
- **Runs uninterrupted**

PROBLEM:
Takes several man-months to construct!

Also: limited to extreme-case analysis

- **Environment = Single-core Processor + Instruction/Data Cache**

Existing Approaches: One-size-fits-all?

- Why construct a SINGLE timing model for ALL programs?
- We are only interested in analyzing a specific program.
- Why not **automatically infer** a **program-specific** timing model?

Outline

- Challenge: Platform (Environment) Modeling
- The GameTime Approach: Learning Program-Specific Environment Model
- Conclusion and Future Directions

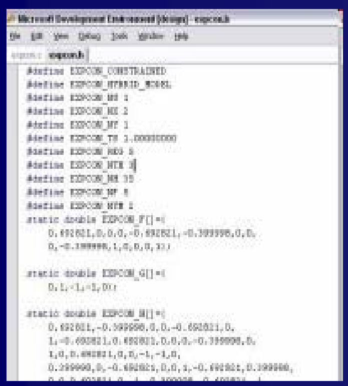
Our Approach and Contributions

[ICCAD '08, ACM TECS]

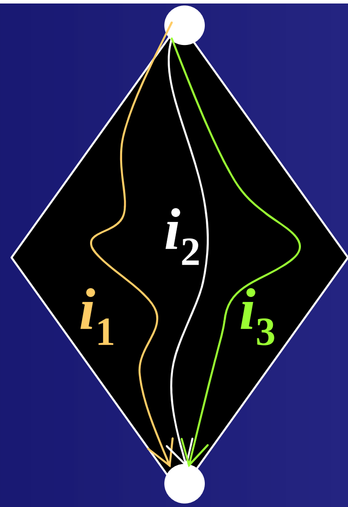
- **Model the estimation problem as a Game**
  - **Tool vs. Platform**
- **Measurement-based, but minimal instrumentation**
  - **Perform *end-to-end* measurements of selected (linearly many) paths on platform**
- **Learn Environment Model**
  - **Automatically *learn* a program-specific model of platform's behavior**
- **Online, randomized algorithm: GameTime**
  - **Theoretical guarantee: can find WCET with arbitrarily high probability under some assumptions**
- **Uses satisfiability modulo theories (SMT) solvers for test generation**

The GameTime Approach: Overview

Game-Theoretic Online Learning + Satisfiability Solving Modulo Theories (SMT)



PROGRAM



CONTROL-FLOW GRAPH

EXTRACT BASIS PATHS

SMT SOLVER GENERATES TEST INPUTS

i_1	42
i_2	75
i_3	101
⋮	⋮

MEASURE EXECUTION TIMES

online

LEARNING ALGORITHM

PREDICT TIMING PROPERTIES
(worst-case, distribution,etc.)

Publication: S. A. Seshia and A. Rakhlin, “Quantitative Analysis of Systems Using Game-Theoretic Learning”, ACM Trans. Embedded Computing Systems.

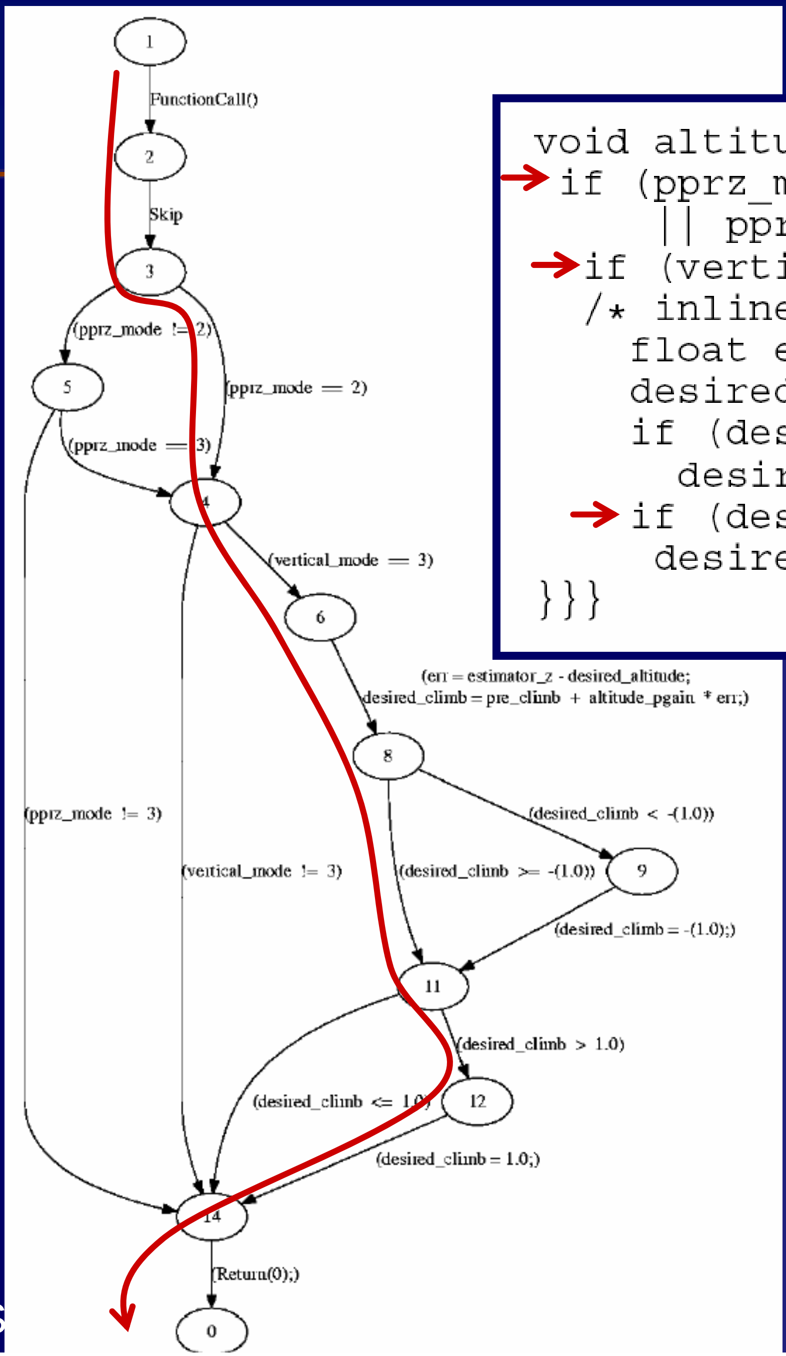
The Game Formulation

- Challenge: Exponentially many program paths and platform states, lack of visibility

- Model as a 2-player Game: Tool vs. Platform
  - Program paths controlled by tool
  - Platform states uncontrollable (controlled by adversary)

- Problems:
  - How to select paths?
  - What is the platform model and how do we learn it?

Search Space = Paths




```
void altitude_control_task(void) {
 if (pprz_mode == PPRZ_MODE_AUTO2
     || pprz_mode == PPRZ_MODE_HOME) {
 if (vertical_mode == VERTICAL_MODE_AUTO_ALT) {
  /* inlined below: function  altitude_pid_run(); */
    float err = estimator_z - desired_altitude;
    desired_climb = pre_climb + altitude_pgain * err;
    if (desired_climb < -CLIMB_MAX)
      desired_climb = -CLIMB_MAX;
    if (desired_climb > CLIMB_MAX)
     desired_climb = CLIMB_MAX;
}}}
```

Must find:

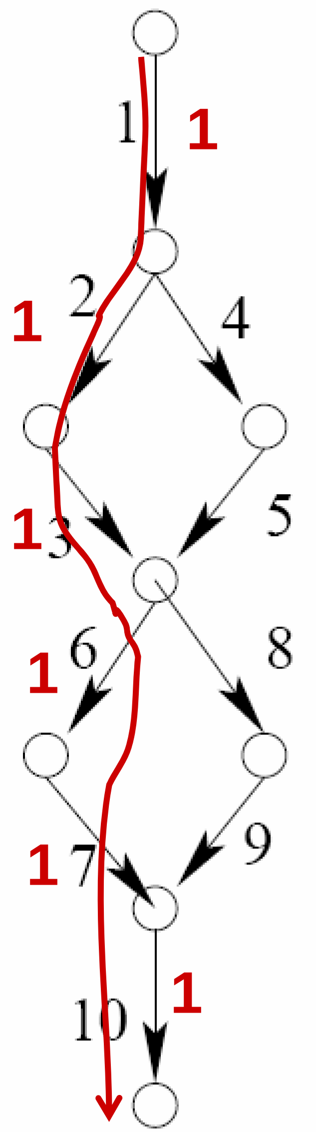
Longest path in the CFG

Only deal with control-dependent timing

Data-dependence: paths → inputs

A Path is a Vector $x \in \{0,1\}^m$

(m = #edges)



$x1 = (1,1,1,0,0,1,1,0,0,1)$

$x2 = (1,0,0,1,1,0,0,1,1,1)$

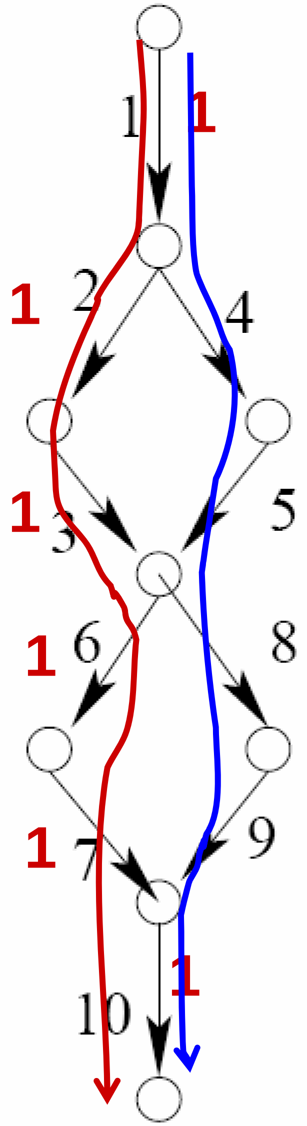
$x3 = (1,1,1,0,0,0,0,1,1,1)$

$x4 = (1,0,0,1,1,1,1,0,0,1)$

<u>Insight:</u>

Only need to sample **a Basis** of the space of paths

Basis Paths



$$x1 = (1,1,1,0,0,1,1,0,0,1)$$

$$x2 = (1,0,0,1,1,0,0,1,1,1)$$

$$x3 = (1,1,1,0,0,0,0,1,1,1)$$

$$x4 = (1,0,0,1,1,1,1,0,0,1)$$

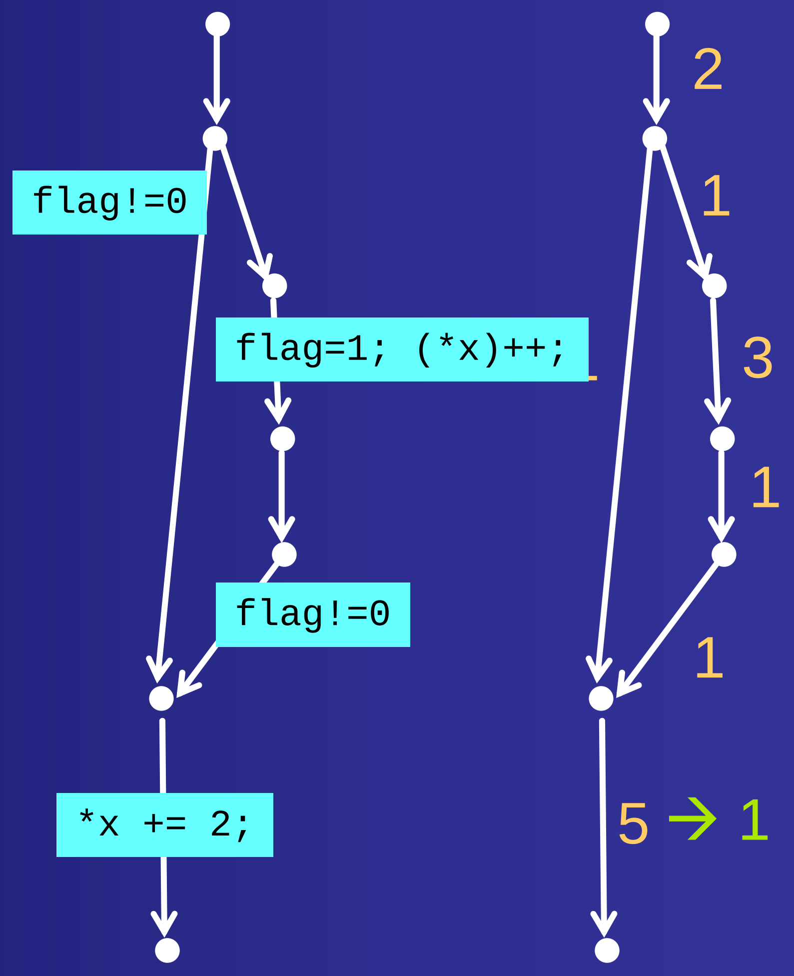
$$x4 = x1 + x2 - x3$$

#(basis paths) $\leq$ m

Useful to compute certain special bases called "barycentric spanners"

Platform Model

- **Adversary picks weights for CFG edges in two stages**
- **Important: weights are path-dependent**



Platform Model

Models path-independent timing

Weights on edges of unrolled CFG

&

Path-specific perturbation

Models path-dependent timing

Formalizing *Repeatable* Timing

Path Dependence

- $\pi = 0$
- $|\pi| \leq \kappa$
- $|E[\pi]| \leq \mu_{max}$

Platform Starting State Dependence

- **w** independent of starting state (too strong!)
- **w** fixed, starting from known state
- **w** selected adversarially (see ACM TECS paper)

Platform Model: Summary

The platform is an adversary picking edge weights

Weights on edges of unrolled CFG $\mathbf{w} \in \mathbb{R}^{\mathbf{m}}$

&

Path-specific perturbation $\pi \in \mathbb{R}^{\mathbf{m}}$

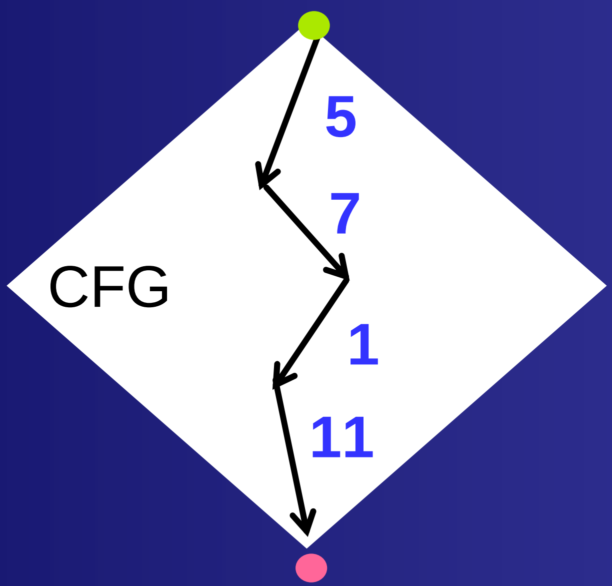
Time taken by path $\mathbf{x}$ is $\mathbf{x} \cdot (\mathbf{w} + \pi)$

Timing Analysis Game (Our Model)

Played over several rounds $t = 1, 2, 3, \ldots, \tau$

At each round t:

Tool picks x_t



Platform picks w_t (= w)

Platform picks $\pi_t(x_t)$

(-1, -1, -1, -1)

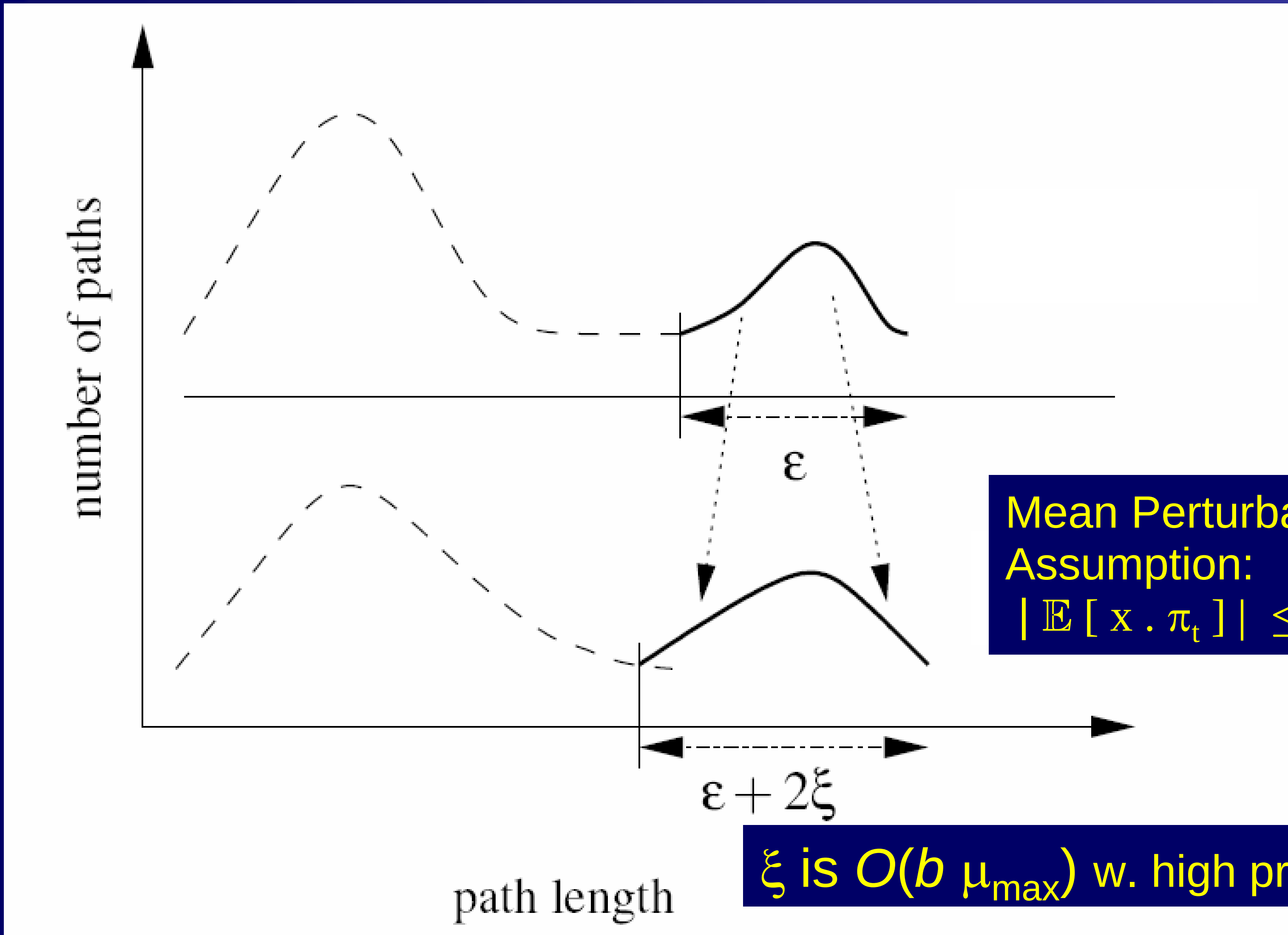
Tool observes $l_t = x_t \cdot (w_t + \pi_t)$

(5+7+1+11) - 4 = 20

At round τ : Tool makes prediction (longest path x^*_τ)

- Tool wins if its prediction is correct

Theorem about Estimating Distribution (pictorial view)



Mean Perturbation Assumption: $\forall x \in \mathit{Paths}$ $|\, \mathbb{E}[\, x \,.\, \pi_t \,] \,| \leq \mu_{max}$

ξ is $O(b\ \mu_{max})$ w. high prob.

GameTime Algorithm: Intuition

- Suppose we knew $w_t + \pi_t$ for all t
- Then, calculate $x^* = \text{argmax}_{x \in Paths} \max_{t=1..\tau} x \cdot (w_t + \pi_t)$

- Idea: **Estimate** $w_t + \pi_t$ to sufficiently high accuracy
- Problem: At any time t, we only see l_t

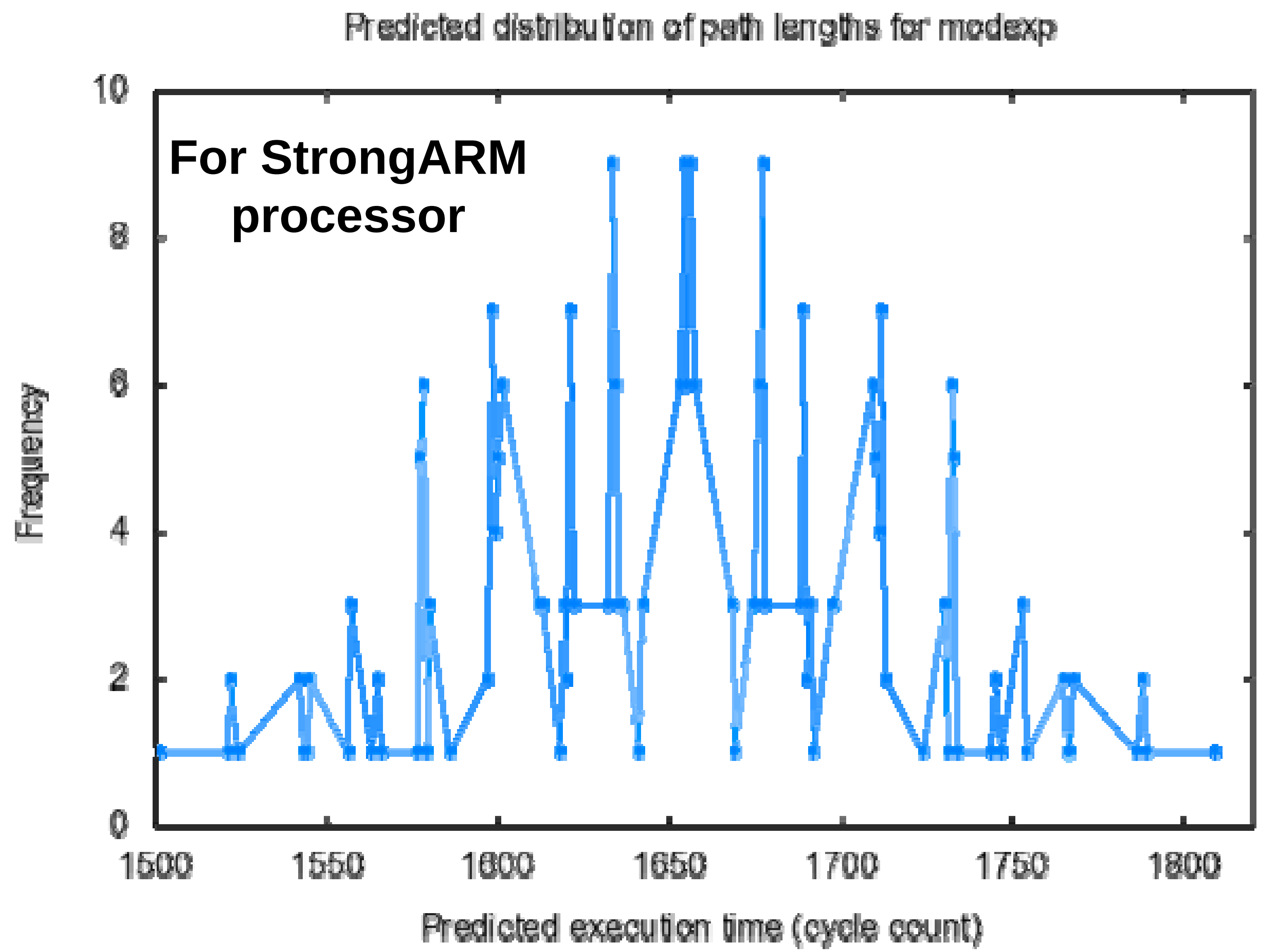
- Two design decisions in GameTime:
  – How to pick x_t ?
    Choose a "basis path" uniformly at random
  – How to estimate $w_t + \pi_t$ from l_t ?
    Perform "least squares estimation"

Summary of Experimental Results

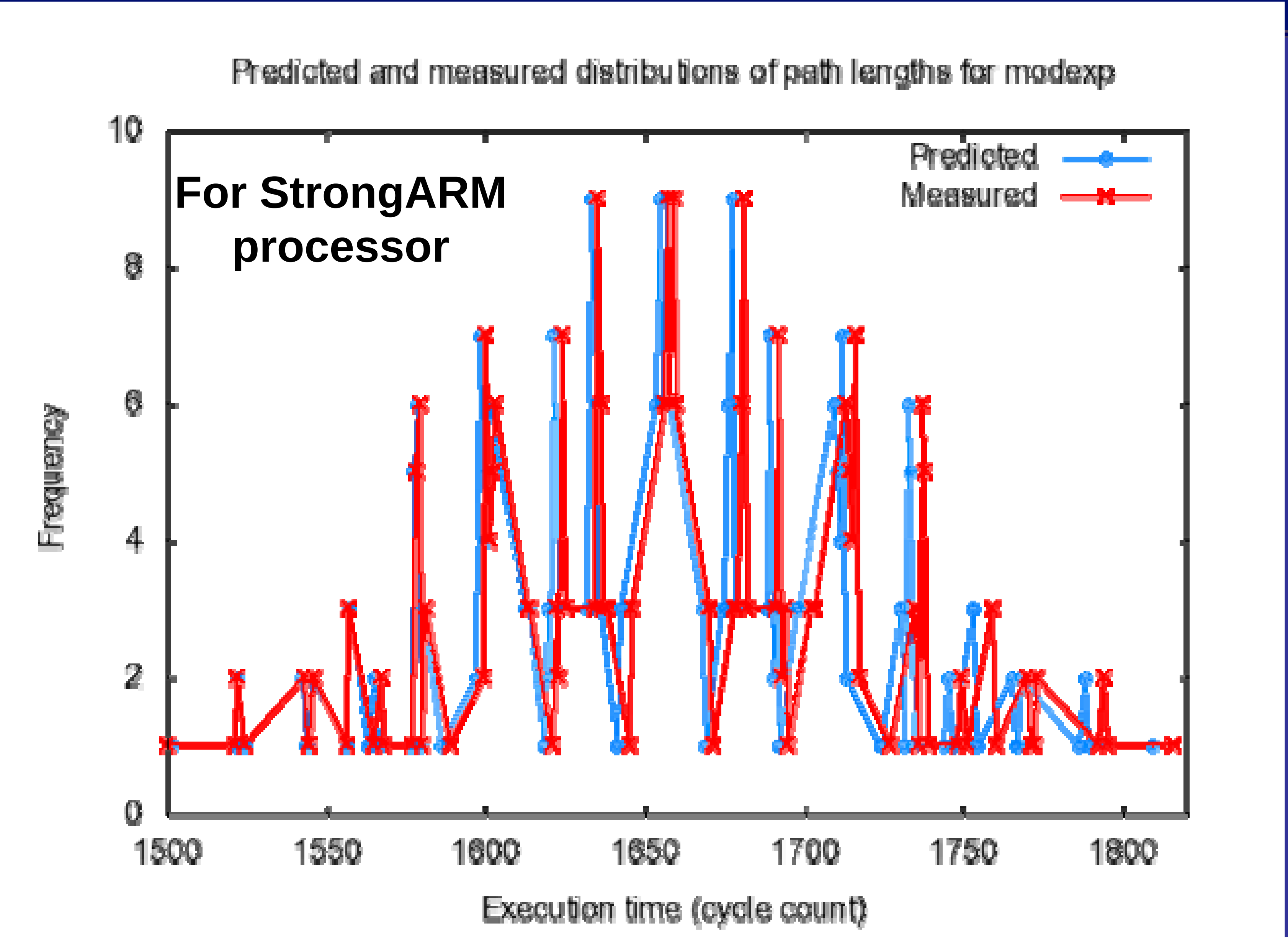
(details in ICCAD'08, ACM TECS papers)

- **GameTime is Efficient**
  - **E.g.: 7 x 10^{16} total paths vs. < 200 basis paths**
- **Sampling basis paths tells us about longer paths we do not sample**
  - **Found paths 25% longer than sampled basis**
- **GameTime can accurately estimate the distribution of execution times with few measurements**
  - **Measure basis paths, predict other paths**
- **GameTime does better than Random Testing**
  - **Found estimates twice as large**
- **GameTime *can even find larger WCET estimates* than conservative WCET estimation tools**

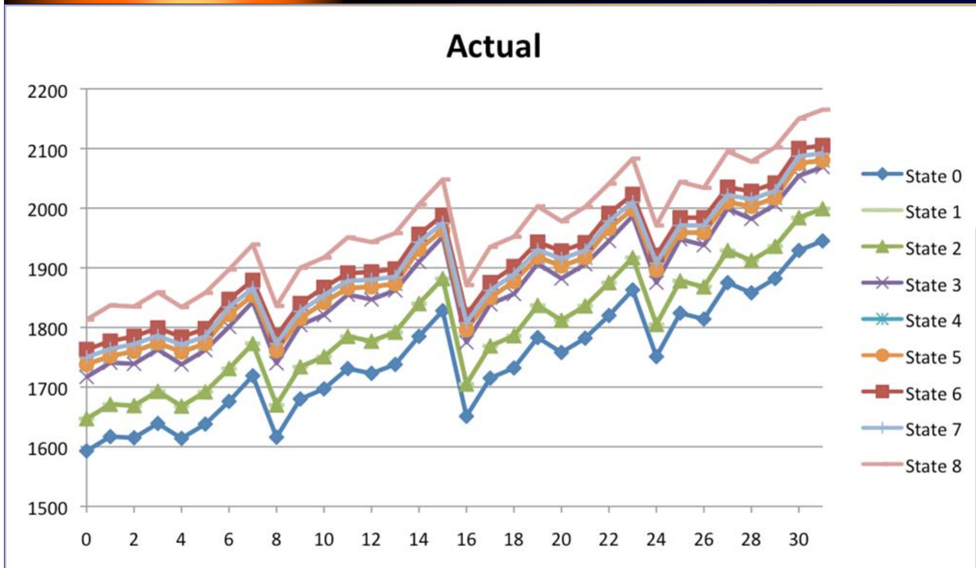
Estimating the Distribution: Modular Exponentiation with 8-bit exponent – predict 256 paths from measuring 9 basis paths

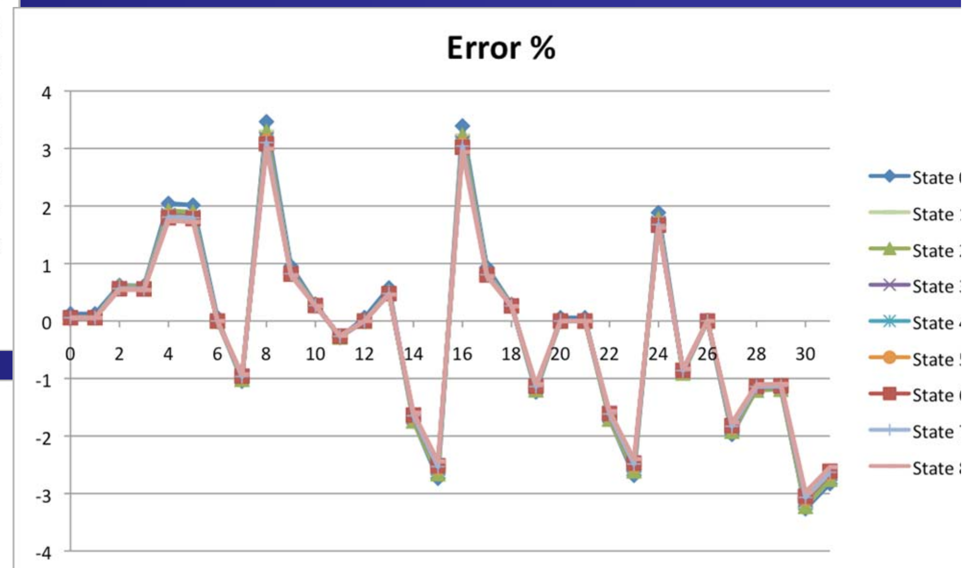


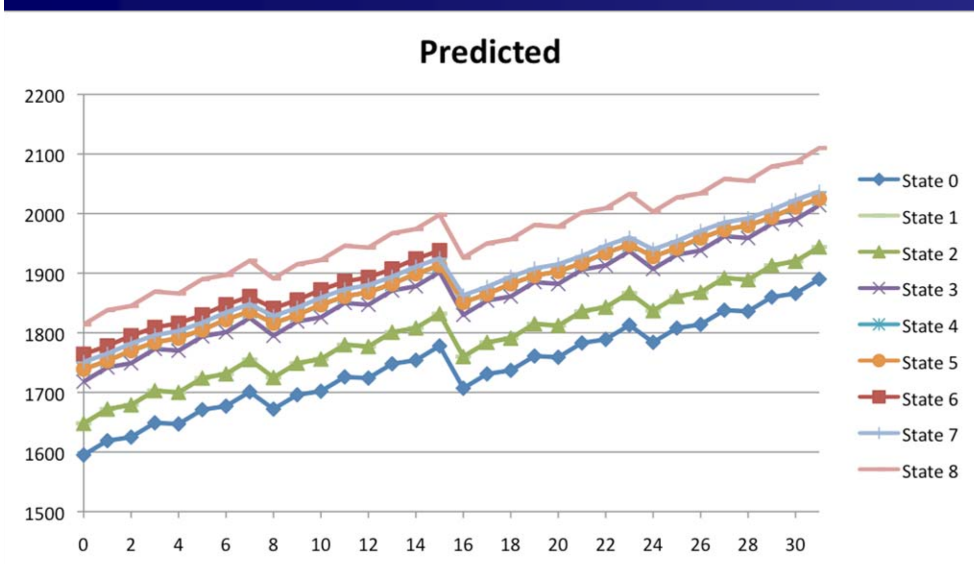
Estimating the Distribution: Modular Exponentiation: predictions in blue, actual 256 measurements in red



GameTime's Accuracy: Different Starting Platform States







$$\text{Error} = \frac{|\text{Actual} - \text{Predicted}|}{\text{Actual}}$$

Conclusions

- Timing analysis important for cyber-physical systems
- Environment modeling is the hard part
  – Current methods too tedious and error-prone
- GameTime: Automatic model generation
  – Active learning from measurements
  – SMT-based basis path testing (a form of coverage)
- Future work
  – Concurrent software: interrupts, multitasking, etc. (see NASA's Toyota UA report)
  – Data-dependent timing
  – Other quantitative analysis problems (e.g. predicting energy consumption)