TECHNISCHE UNIVERSITÄT DRESDEN

INVESTIGATING THE LIMITATIONS OF PVF FOR REALISTIC PROGRAM VULNERABILITY ASSESSMENT

Björn Döbel (TU Dresden)
Horst Schirmeier (TU Dortmund)
Michael Engel (TU Dortmund)

Berlin, 21.01.2013

Motivation

Software development today:

- How fast does my program run?
- How much energy does it consume?
- How many software errors does it contain?

Software development in the future:

- Which functions are most vulnerable against certain hardware errors?
- Should I use implementation A or B of a certain algorithm?
- Which impact will compiler optimizations have on my program's vulnerability?

Vulnerability Analysis Today

Hardware vulnerability analysis: Architectural Vulnerability Factor (AVF)[1]

- Established technique
- Requires both application and hardware model
- Hardware model out of reach in practical SW development

[1] Mukherjee et al.: *A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor*, IEEE MICRO 2003

[2] Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009

Vulnerability Analysis Today

Hardware vulnerability analysis: Architectural Vulnerability Factor (AVF)[1]

- Established technique
- Requires both application and hardware model
- Hardware model out of reach in practical SW development

Software-only analysis: Program Vulnerability Factor (PVF)[2]

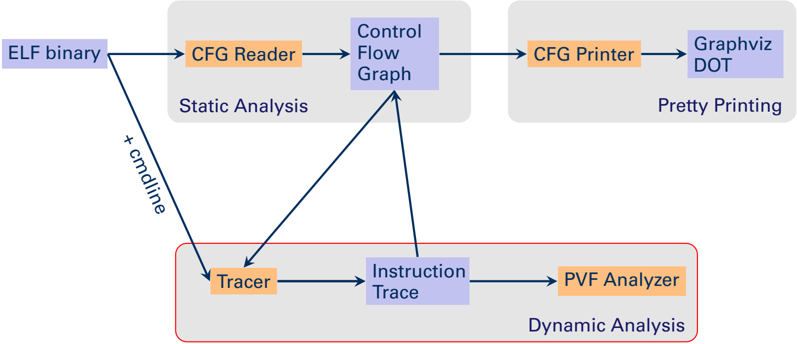
- Replace hardware with an abstract model
- Requires application knowledge only

---

[1] Mukherjee et al.: *A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor*, IEEE MICRO 2003

[2] Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009

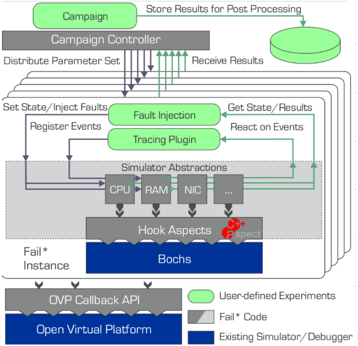
PVF Analysis for x86



Source code: `https://github.com/TUD-OS/PVFAnalyzer`

FAIL/*: Fault Injection

- Framework for performing FI campaigns[3]
- Interchangeable hardware model: Bochs/x86
- Benefit: real distribution of errors for a given platform



[3] H. Schirmeier, M. Hoffmann, R. Kapitza, D. Lohmann, and O. Spinczyk: FAIL/*: *Towards a versatile fault-injection experiment framework*, ARCS 2012

Experiment Setup

Application:

- `nanojpeg` JPEG decoder
- Decode a small JPEG image into bitmap
- GCC 4.4.5, fully optimized $\rightarrow$ 3.7 million instructions

Fault Injection:

- FI campaign using FAIL/*
- Flip every bit in every used register once and record outcome
- Runtime: about a week on fairly large cluster

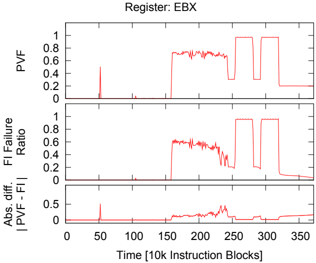
PVF Analysis:

- Analysis of a single trace
- Same fault model as for FI
- Runtime: a couple of minutes on my laptop

TECHNISCHE UNIVERSITÄT DRESDEN

Initial Results

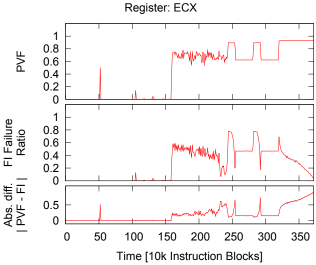
- Program run split into blocks of 10,000 instructions
- For each block: compare computed PVF and fault ratio from FI experiments
- Question: What do we consider a fault?
  - Program crashes
  - Infinite loops
  - Silent data corruption



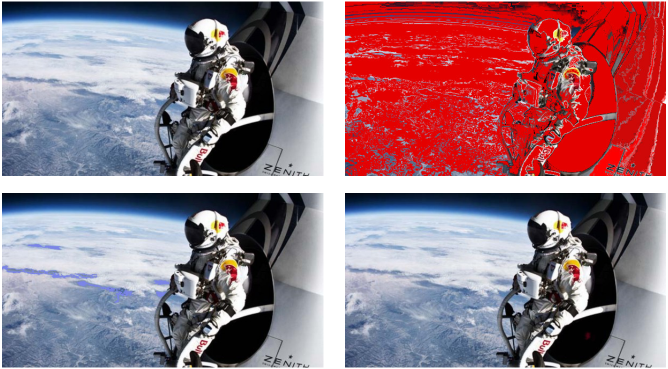
Observation #1: PVF is a fairly good prediction of the fault rate if we consider all SDC to be faults.

Understanding PVF-FI Deviations

1. FI works on bit granularity, PVF analyses registers
2. PVF does not analyze bit masking etc. operations
3. PVF analysis stops on block boundaries – no propagation of fault state for more than 10,000 instructions



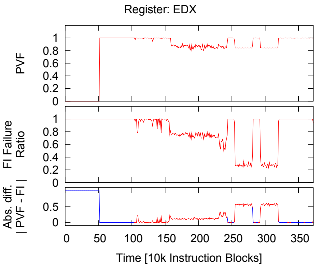
Corrupted data?



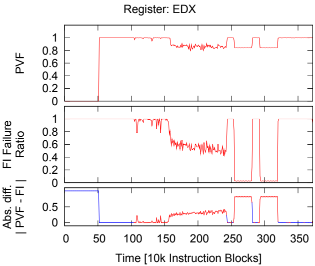
Observation #2: Not all data corruption can be considered equally faulty.

Considering Data Quality

All SDC are failures:



PSNR threshold = 20:



Incorporating Quality Into PVF?

- Similar results for an analysis of H.264 video decoding: control data (high fault rate) vs. payload data (low fault rate)[4]
- Compiler extension:
  - `reliable` and `unreliable` data modifiers
  - Special protection (encoding / redundancy) for `reliable` data

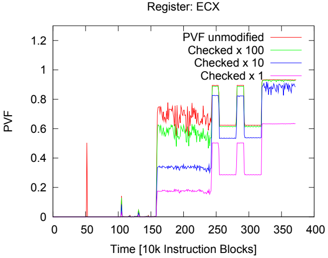
Observation #3: Using reliability annotations, the compiler can determine the reliability level of a hardware resource and propagate this information to a PVF analyzer.

[4] A. Heinig, M. Engel, F. Schmoll, and P. Marwedel: *Improving transient memory fault resilience of an H.264 decoder*, STMEDIA 2010

Modeling the Effect of FT methods

- $PVF(app)$ models probability of a fault affecting execution
- Replicated execution (e.g., RMT): tolerates faults $\frac{n}{2} - 1$ independent faults

  $PVF_{rep}(app) := PVF(app)^{\frac{n}{2}}$
- Encoded processing:
  - Checksummed data, periodic checking
  - Limit PVF analysis to smaller periods



Now What?

PVF seems promising, but is not there yet.

- We need to incorporate data quality information.
- Compiler support: propagate resource vulnerability information to PVF analyzer.
- Analysis of different fault models
  - Original PVF: register and ALU errors
  - How about memory bit flips, decoding errors, ...?

PVF will never fully replace full-featured FI campaigns.

- Analysis of complex applications and fault models
- Strong limitation: Abstract hardware model