VarEMU: An Emulation Testbed for Variability-Aware Software

Lucas Wanner, Salma Elmalaki, Liangzhen Lai, Puneet Gupta, and Mani Srivastava

wanner@ucla.edu
The Variability Problem

Nominally identical parts, different characteristics

Frequency  Errors  Power

Variability (%)

Year


Total Power
Static (Sleep) Power
Variability-Aware Software

Software

App

App

OS

Sense and Expose
Variability-Aware Software

Software

- App
- App
- OS

Activation Control

HW/SW Parameters

App Code

Sense and Expose

Quality

Problem: hardware is not instrumented for variability sensing
### Motivation

How to **Sense and Expose** Variations to Software?

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Observability</th>
<th>Control</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrumented Hardware</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW-In-The Loop Sim</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gate/RTL-Level Sim</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>VarEMU</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Architecture

QEMU

Virtual Machine Monitor
 Runs unmodified SW images
 Cross-architecture support
 Binary translation (fast)
Architecture

VarEMU

QEMU Core
Power + Aging Model
Fault Framework
Energy + Time Accounting

Query energy
Control faults
Suffer faults

Configure, change, query

User + Software Monitor

Virtual Machine
Cycle and Time Accounting

Translation time

add r1, r2, r3

Translated ops
- x
- y
- z

cycle counting

Execution time

# cycles
class
fault status

Special case: WFI (standby instruction): Start sleep interval until next interrupt
VarEMU: An Emulation Testbed for Variability-Aware Software

Virtual Machine

VarEMU

Instruction disassembly & translation

Cycle & Time Accounting

Aging Model

Power Model

User + Software Monitor
Aging and Power Model

Frequency
Voltage
Temperature

Parameters for each (class of) instructions
Active + Sleep
Power and Aging Model

technology +
design
dependent

change due to
aging & user
input

Power = f (current state)
Power Model

- Best Case
- Worst Case
- Nominal Case

Sleep Power (µW) vs. Temperature (°C)

Normalized to match 45nm Cortex-M3 test chip
Aging Model

\[ \frac{\partial V_t}{\partial t} (V) \]

\[ \begin{align*}
\text{0.01} & \quad \text{0.02} \\
0 & \quad 7500000 \\
15000000 & \quad 22500000 \\
30000000 & \quad 60^\circ \text{C, reference (r) vs. model (m)}
\end{align*} \]

\[ \begin{align*}
\text{0.8V (r)} & \quad \text{1V (r)} & \quad \text{1.2V (r)} \\
0.8V (m) & \quad 1V (m) & \quad 1.2V (m)
\end{align*} \]
VarEMU: An Emulation Testbed for Variability-Aware Software

Virtual Machine

VarEMU

Energy = Time × Power

User + Software Monitor

Instruction disassembly & translation

Cycle & Time Accounting

Energy Accounting

Aging Model

Power Model
VarEMU: An Emulation Testbed for Variability-Aware Software
Faults

Translation time

add r1, r2, r3

fault status

Execution time

Translated ops

\begin{align*}
x \\
y \\
z
\end{align*}

cycle counting
Faults

Translation time

add r1, r2, r3

fault status

Execution time

pre

Translated ops
x
y
z

replace

post

cycle counting
Fault Module: translation Time

Instruction marked as faulty?

No: generate standard code
mul rd, rn, rs

Yes: generate two code paths
fault module
mul rd, rn, rs
Fault Module: execution time

Translated instruction

Tests

Faulty version

Standard (correct) version
Fault Module: execution time

Translated instruction
- Tests
  - Faulty version
  - Standard (correct) version

Input: architectural state, instruction info
- Privileged mode?
- Faults enabled in the VM?
- User-defined conditions
- Chose between correct and faulty version
Fault Module: execution time

Translated instruction

Tests

Faulty version

Standard (correct) version

Example: MUL + stuck at one

```c
env->regs[rd] = env->regs[rm] * env->regs[rs] | 0x01;
```
**VarEMU: An Emulation Testbed for Variability-Aware Software**

#### Virtual Machine

- Instruction disassembly & translation
- Cycle & Time Accounting

#### VarEMU

- Energy Accounting
- Aging Model
- Power Model
- Fault Model

#### User + Software Monitor

- Virtual Hardware Device
VarEMU: An Emulation Testbed for Variability-Aware Software
VM-VarEMU interaction

- VM interacts with VarEMU through memory mapped registers
- Commands
  - *read* command creates a checkpoint for all VarEMU registers
  - *faults* command propagates value to fault module
    - zero: faults disabled
    - not zero: implementation-dependent, typically means faults *may* now occur
- Multiple registers for time, energy, cycles
- Linux driver handles interaction through syscall interface and provides per-process energy accounting
VarEMU Linux support

read registers
save error status

process

read registers
restore error status

VarEMU status
is part of process context

kernel
App example

```c
int main() {
    vemu_regs regs; ...
    syscall(VAR_SYSCALL_READ, READ_PROC, &regs);
    printf("active energy: %d\n", regs.total_act_energy);
    int fault_status = 1;
    syscall(VAR_SYSCALL_WRITE, FAULT_ST, &fault_status);
    x = y * z;
    fault_status = 0;
    syscall(VAR_SYSCALL_WRITE, FAULT_ST, &fault_status);
}
```
VarEMU: An Emulation Testbed for Variability-Aware Software

- Virtual Machine
- Instruction disassembly & translation
- Cycle & Time Accounting
- I/O
- User + Software Monitor
- VarEMU
- Energy Accounting
- Aging Model
- Power Model
- Fault Model
- Virtual Hardware Device
- Variability Expedition
Dynamic monitoring and control

- Socket interface (QEMU Machine Protocol)

- VarEMU status queries (energy, error status, ...)

- VarEMU power model parameter updates
  
  - Example: change parameter zero (temperature) of class zero (all) to 60 (°C)
Evaluation: time accounting accuracy

Hardware timing vs. VarEMU

- empty loop
- matrix multiplication
- auto correlation

Accuracy (%) vs. Time

Accuracy (%)

100%
99%
98%
97%
96%
95%

0 50 100 150 200

Time
Evaluation: overhead
Case study: approximate arithmetic

- Approximate adder using partial sums [Kahng and Kang, DAC’12]
  - Fault implementation with variable accuracy: 25% accuracy means that 25% of the partial sums are corrected
  - Clock period reduced by 25%, with 6% recovery overhead

97% of pixels within 5% of “correct” version
Case Study: Dynamic Reliability Management

- Higher voltage is needed to support same frequency as the device ages
  - Non-adaptive voltage, margined for the worst-case
  - Adaptive voltage, dynamically tailored to match aging process

11% energy savings
Research and Education using VarEMU

- Application robustness to intermittent and permanent faults [Sharma et. al., ICCD’13]
  - Fault model using VarEMU + ModelSim
  - Code transformations (e.g. operand swap) to increase robustness to faults
- Graduate-level courses at UCLA
  - Battery models
  - Fault experiments
- Variability-Aware Real-Time OS [Martin, et. al., submitted to TECS]
  - Learn power characteristics for each instance, tailor task activations and quality to achieve desired lifetime

![Graph showing Instance and Temperature Profile (Inst/Temp) vs. Velocity Estimate Error (RMSE). The graph includes data points for B/B, B/N, B/W, N/B, N/N, N/W, W/B, W/N, W/W, with markers indicating Assumed Worst Case and With Instance Modeling.]
Final remarks

- VarEMU: a tool to evaluate variability-aware software
  - Variability-aware aging and power models
  - Energy, cycle accounting
  - Fault framework
  - Linux + embedded software stack
    - Per-process accounting
    - Fine-grained control over faults
- Limitations
  - Cycle counting: simple model doesn’t take pipeline hazards, branch prediction misses, cache, etc. into account
  - Fault framework: only architecturally visible states can be altered
    - External tools may be used to simulate more complex faults
- Availability:
  - VarEMU is open software. github.com/nesl/varemu
  - NSF Variability Expeditions: variability.org
VarEMU: An Emulation Testbed for Variability-Aware Software

Lucas Wanner, Salma Elmalaki, Liangzhen Lai, Puneet Gupta, and Mani Srivastava

wanner@ucla.edu