ViRUS: Virtual Function Replacement Under Stress

Lucas Wanner and Mani Srivastava
wanner@ucla.edu, mbs@ucla.edu
Networked & Embedded Systems Lab
UCLA
Polymorphic Engines

• Change the code, keep the semantics, hide the malware

```assembly
movi r1, 0
addi r1, 5
movi r1, 7
subi r1, 2
```

Our work: change the code, keep the semantics but change the quality, use energy more efficiently
Algorithmic Choice

Given multiple alternatives for a function $f$, dynamically select one version depending on the variable execution context.
Energy stress from process variations

**Intel Core i5: 12-17%**
Balaji, HotPower, '12

**DDR3: 32%**
Gottscho, ESL 4(2), '12

**SAM3U Sleep: 14x**
Energy stress from context
Design considerations for algorithmic choice

```
<table>
<thead>
<tr>
<th>f0</th>
<th>f1</th>
<th>f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>
```

- **App**: Functionality layer
- **Tests**: Verification layer
- **Sensors**: Input data
- **OS**: Operating system layer
Design considerations for algorithmic choice

![Diagram showing the relationship between App, f, f₀, f₁, f₂, Sensors, and OS]

- App
- f
- f₀, f₁, f₂
- Sensors
- OS

Diagram illustrates the flow from Sensors to the application, passing through functions f₀, f₁, f₂, and ultimately connecting to the OS.
Design considerations for algorithmic choice

- How to expose $f_0$, $f_1$, $f_n$ as $f$
- How to capture programmer intent
- How to trigger adaptation

App

$f_0$  $f_1$  $f_2$

Sensors  OS
Exposing multiple versions of a function

\[ f_0 \]

\[ f_1 \]

\[ f_2 \]
Exposing multiple versions of a function

*f

\[ f_0 \]

\[ f_1 \]

\[ f_2 \]

\[ \text{double } (*v\_log)( \text{ double args }) ; \]

\[ \text{F\_ONE(log, 0, double, double, log)} \]

\[ \text{F\_ONE(log, 1, double, double, logf)} \]

\[ \text{LEVELS(log, 3)} \]

\[ \text{LIB\_INIT(log)} \]

f is a knob, registered on app initialization
Capturing programmer intent

adaptation policy

<table>
<thead>
<tr>
<th>function</th>
<th>priority</th>
<th>sensor</th>
<th>range</th>
<th>quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>temperature</td>
<td>[0, 40)</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>temperature</td>
<td>[40, 100)</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>battery</td>
<td>[20, 100)</td>
<td>{0, 1, 2}</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>battery</td>
<td>[0, 20)</td>
<td>{2}</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>temperature</td>
<td>[0, 60)</td>
<td>{0}</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>temperature</td>
<td>[60, 100)</td>
<td>{1, 2}</td>
</tr>
</tbody>
</table>

config file parsed on app initialization
## Capturing programmer intent

<table>
<thead>
<tr>
<th>function</th>
<th>priority</th>
<th>sensor</th>
<th>range</th>
<th>quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>temperature</td>
<td>[0, 40)</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>temperature</td>
<td>[40, 100)</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>battery</td>
<td>[0, 20)</td>
<td>{2}</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>temperature</td>
<td>[0, 60)</td>
<td>{0}</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>temperature</td>
<td>[60, 100)</td>
<td>{1, 2}</td>
</tr>
</tbody>
</table>

**T=45, B=50**

### knob table

<table>
<thead>
<tr>
<th>sensors</th>
</tr>
</thead>
</table>

### adaptation policy

Capturing programmer intent
Triggering adaptation

OS

Energy Stress Sensors

app

libx

config

control

policy

knobs

adaptation

f

g

h
Triggering adaptation

- **app**
- **libx**
- **config**
- **control**
  - policy
  - knobs
  - adaptation
- **monitor library**
  - sig handler
  - constructor
- **Stress Daemon**
  - sensors
  - values
- **socket**

Diagram connections:
- app → libx
- config → control
- monitor library → sig handler
- monitor library → constructor
- Stress Daemon → sensors values
- socket → Stress Daemon
- alarm → register process

Legend:
- OS
- constructor
- sig handler
- register process
- alarm
- sensor values
- socket
Evaluation setup: VarEMU Emulator

Virtual Machine

- Instruction disassembly & translation
- Cycle & Time Accounting

VarEMU
- Energy Accounting

User + Software Monitor

Virtual Hardware Device
- Power Model
Evaluation setup: VarEMU Emulator

- VM interacts with VarEMU through memory mapped registers
- Multiple registers for time, energy, cycles
- Linux driver handles interaction through syscall interface and provides per-process energy accounting
- A small library linked with each test application dumps energy and time status after each run is completed
Runtime overheads

on initialization

Time (μs)

socket
gerger functions
discover sensors
parse adaptation policy

1GHz frequency, 10 functions, 10 rules

180 \times \text{double exp}
Runtime overheads on initialization

Time (μs)

0  100  200  300  400  500  600  700
Runtime overheads

on initialization

19 × double exp

periodic

file I/O
Memory overhead of multiple versions

- **exp**: Memory usage for double precision is significantly higher than for single precision and fast versions.
- **log**: Similar to exp, double precision consumes more memory.
- **pow**: Double precision and single precision have comparable memory usage, with fast and faster versions being the least memory-intensive.
- **sin**: Double precision usage is again higher, with single precision being a close second.
- **cos**: Double precision has the highest memory usage, followed by single precision.
- **tan**: Double precision has the highest memory usage, with single precision being considerably less.

The diagram shows the memory usage in KB for different versions of these mathematical functions.
Memory overhead of multiple versions

Memory Usage (KB)

- exp
- log
- pow
- sin
- cos
- tan
- lgamma

10% overhead
Energy usage across versions

- blackscholes: 50%
- swaptions: 30%
- whetstone: 55%
Quality: swaptions

NRMSE: 0.5%
MAPE: 3.8%
Quality: blackscholes

NRMSE: 4.2%
MAPE: 40%
Energy and quality for different policies: Long-running blackscholes

Lifetime: 1 year
Speedup 3600:1
Temperature Profile: Stovepipe Wells, CA, 2009

Quality = High

MAPE (%)

1E+00
1E-02
1E-04
1E-06

Total Energy (W-h)

Best
Nominal
Worst
Energy and quality for different policies: Long-running blackscholes

MAPE (%)

Total Energy (W-h)

Best
Nominal
Worst

Quality = Low

36% energy savings
(vs. 40% without sys overhead)

Quality = High
Energy and quality for different policies: Long-running blackscholes

Constant quality across instances

11% savings
Energy and quality for different policies: Long-running blackscholes

![Graph showing MAPE vs Total Energy for different policies. The x-axis represents Total Energy (W-h) ranging from 15 to 28, and the y-axis represents MAPE (%) ranging from 1E-06 to 1E+00. The graph compares Best, Nominal, and Worst policies. The Best policy is represented by green circles, the Nominal by yellow circles, and the Worst by red circles. The graph shows that as Total Energy increases, MAPE generally decreases.]
Energy and quality for different policies: Long-running blackscholes

2x range in quality, 12-20% savings

Power policy

30 to 100 mW: Low
25 to 30 mW: Med
0 to 25 mW: High
Summary

• Runtime support systems can help embedded applications adapt to energy stress, trading off quality and energy cost

• ViRUS: Virtual Function Replacement Under Stress
  • Control+Monitoring Architecture for Algorithmic choice + Polymorphic math library: negligible runtime overhead, 10% memory
  • 30-55% energy savings, up to 4% degradation in quality
  • Code and data available at github.com/nesl and variability.org

• Looking forward
  • Automated / suggested adaptation policies
  • Higher level view of context / energy stress
    • e.g., elevated energy consumption; little battery remaining
  • Android/Java extension: CAreDroid
    • + cloud offloading
ViRUS: Virtual Function Replacement Under Stress

Lucas Wanner and Mani Srivastava
wanner@ucla.edu, mbs@ucla.edu
Networked & Embedded Systems Lab
UCLA