Bending microarchitectural weird machines towards practicality

Ping-Lun Wang, Riccardo Paccagnella, Riad S. Wahby, and Fraser Brown
USENIX Security, 08/14/2024

Carnegie Mellon University
CyLab
Security and Privacy Institute

~~A new µArch side-channel~~ µArch computer?

µArch weird machines (µWMs)

▲ CPU

µWMs: difficult to analyze!

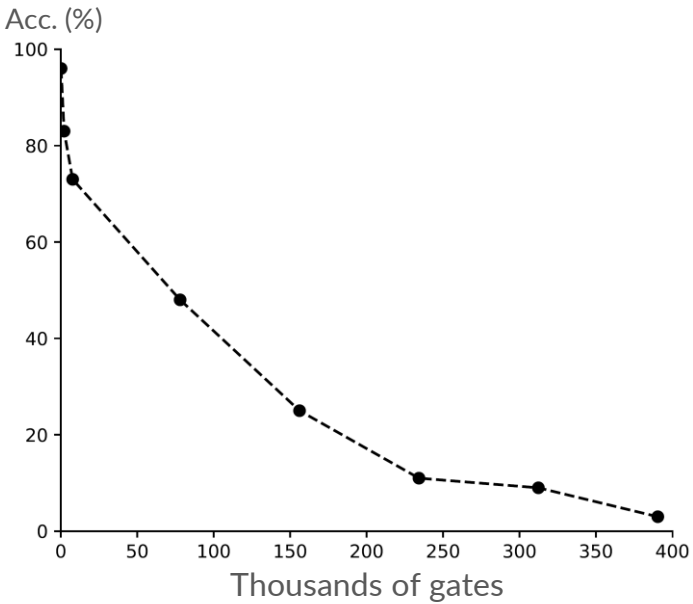
- 😎 µArch states → invisible to Arch
- 🤯 Prevents code analysis
- 😈 Defeats debuggers and emulators

➡ Seems to be great for program obfuscation?

Credit: irasutoya.com

Existing µWMs need improvements…

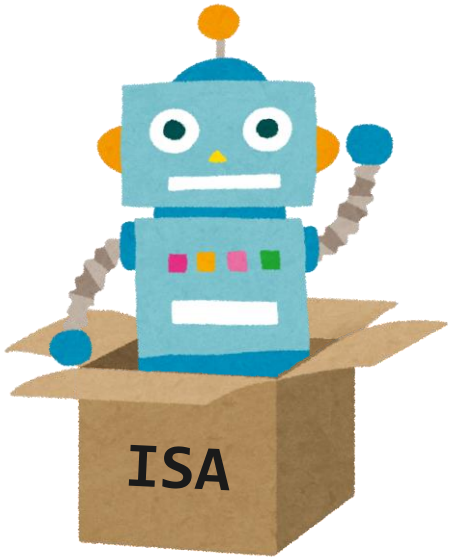
- Not scalable for large computations



- Difficult to create new µWMs

```
_32BitValueToCacheState(C_arch, C1, NULL);
_32BitValueToCacheState(D_arch, D1, NULL);
_32BitValueToCacheState(E_arch, E, NULL);
_32BitValueToCacheState(K[0], K0, NULL);
_32BitValueToCacheState(W_arch, W, NULL);
LeftShift(A1, A1, 27);
mfence();
lfence();
trash = _32BitAnd(B1, C1, B_AND_C, trash);
mfence();
lfence();
trash = _32BitNot(B2, NOT_B, trash);
mfence();
lfence();
trash = _32BitAnd(NOT_B, D1, NOT_B_AND_D, trash);
… // 128 lines in total
```

Flexo: a new design for µWM



✨ New circuit construction!

- 4× smaller, 25× faster, ≥ 97.8% accuracy

✨ Compiler for µWMs!

- C/C++ to µWMs

✨ µWM binary packer!

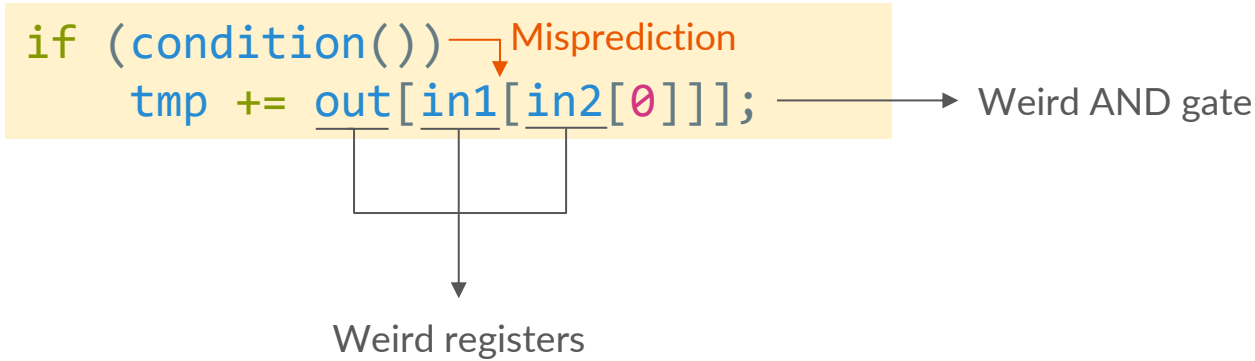
- Obfuscate UPX with AES and Simon

Credit: irasutoya.com

Arch vs. µArch “weird” computation

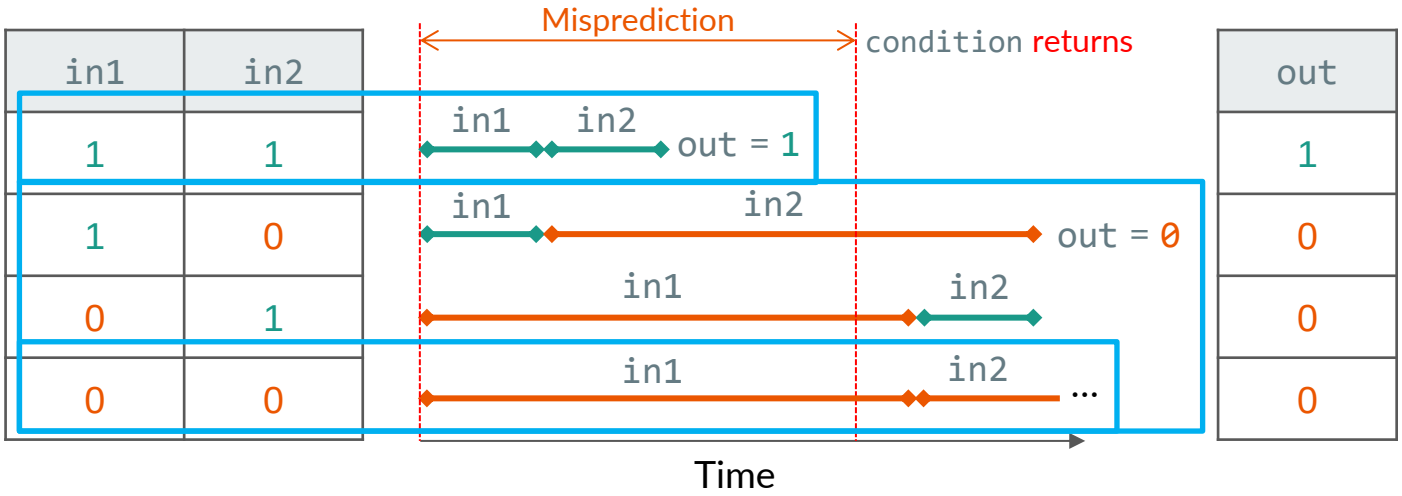
	“Normal” computation	“Weird” computation
How to compute	ISA instructions	**“Weird” gates**: memory operations during transient execution
Where values are stored	Registers or memory	**“Weird” registers**: cache residency information - In the cache: value 1 - Out of cache: value 0

The construction of a weird gate

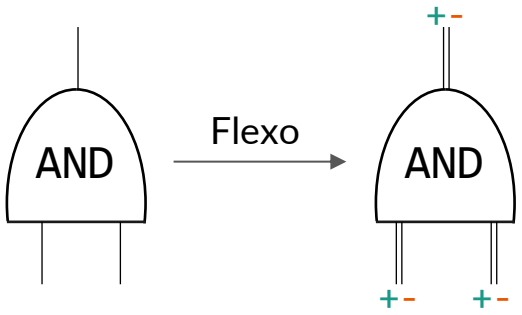


Prior work's AND gate

```
if (condition())
    tmp += out[in1[in2[0]]];
```
Misprediction



Flexo's circuit construction: differential encoding



- Use two wires for one binary value
  - 1 → (+, -) = (1, 0)
  - 0 → (+, -) = (0, 1)
  - Invalid states: (1, 1) or (0, 0)
  - **Provides a lot of benefits!**

New encoding → much more scalable gates!

in1	in2	out
1	0	1
0	1	1
1	1	0
0	0	0

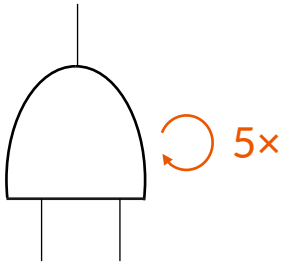
```
if (condition()) {
    tmp += out_p[in1_p[0] + in2_m[0]];
    tmp += out_p[in1_m[0] + in2_p[0]];
    tmp += out_m[in1_p[0] + in2_p[0]];
    tmp += out_m[in1_m[0] + in2_m[0]];
}
```

OR gates

AND gates

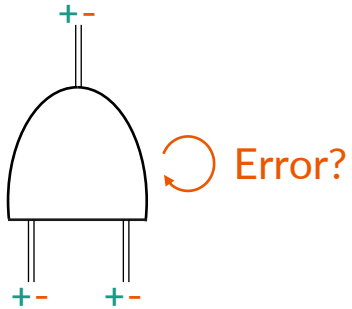
Any gate with ≤ 4 inputs works!!
E.g., 4-bit XOR or 2-bit adder

Error correction with dynamic voting



Prior work (GoT) [1]: 3-out-of-5 voting

- 💥 Fixed 5× overhead (81% → 96%)
- 💥 Low efficacy (57% → 83%)



Flexo: rerun only when error occurs

- ✨ Dynamic overhead (1.24× to fix 81% → 100%)
- ✨ High efficacy (0.3% → 99.9%)

[1] The Gates of Time: Improving Cache Attacks with Transient Execution, Katzman et al., USENIX Sec '23

Flexo's compiler: the first compiler for μWMs

```
_32BitValueToCacheState(C_arch, C1, NULL);
_32BitValueToCacheState(D_arch, D1, NULL);
_32BitValueToCacheState(E_arch, E, NULL);
_32BitValueToCacheState(K[0], K0, NULL);
_32BitValueToCacheState(W_arch, W, NULL);
LeftShift(A1, A1, 27);
mfence();
lfence();
trash = _32BitAnd(B1, C1, B_AND_C, trash);
mfence();
lfence();
trash = _32BitNot(B2, NOT_B, trash);
mfence();
lfence();
trash = _32BitAnd(NOT_B, D1, NOT_B_AND_D, trash);
mfence();
lfence();
trash = _32BitOr(B_AND_C, NOT_B_AND_D,
B_AND_C_OR_NOT_B_AND_D, trash);
mfence();
lfence();
trash = _32BitAdder_impl(A1, B_AND_C_OR_NOT_B_AND_D,
SUM_1, WORDSIZE, trash);
mfence();
lfence();
…
```

(128 lines in total)

```
void __weird__sha1_round(
    unsigned* input,
    unsigned* output, unsigned* error_output
) {
    unsigned a = input[0];
    unsigned b = input[1];
    unsigned c = input[2];
    unsigned d = input[3];
    unsigned e = input[4];
    unsigned w = input[5];
    unsigned k = 0x5A827999;

    unsigned f = (b & c) | ((~b) & d);
    unsigned temp = ROL(a, 5) + f + e + w + k;

    output[0] = temp;
    output[1] = a;
    output[2] = ROL(b, 30);
    output[3] = c;
    output[4] = d;
}
```

(21 lines in total)

😇 No low-level details

✨ Optimizations!

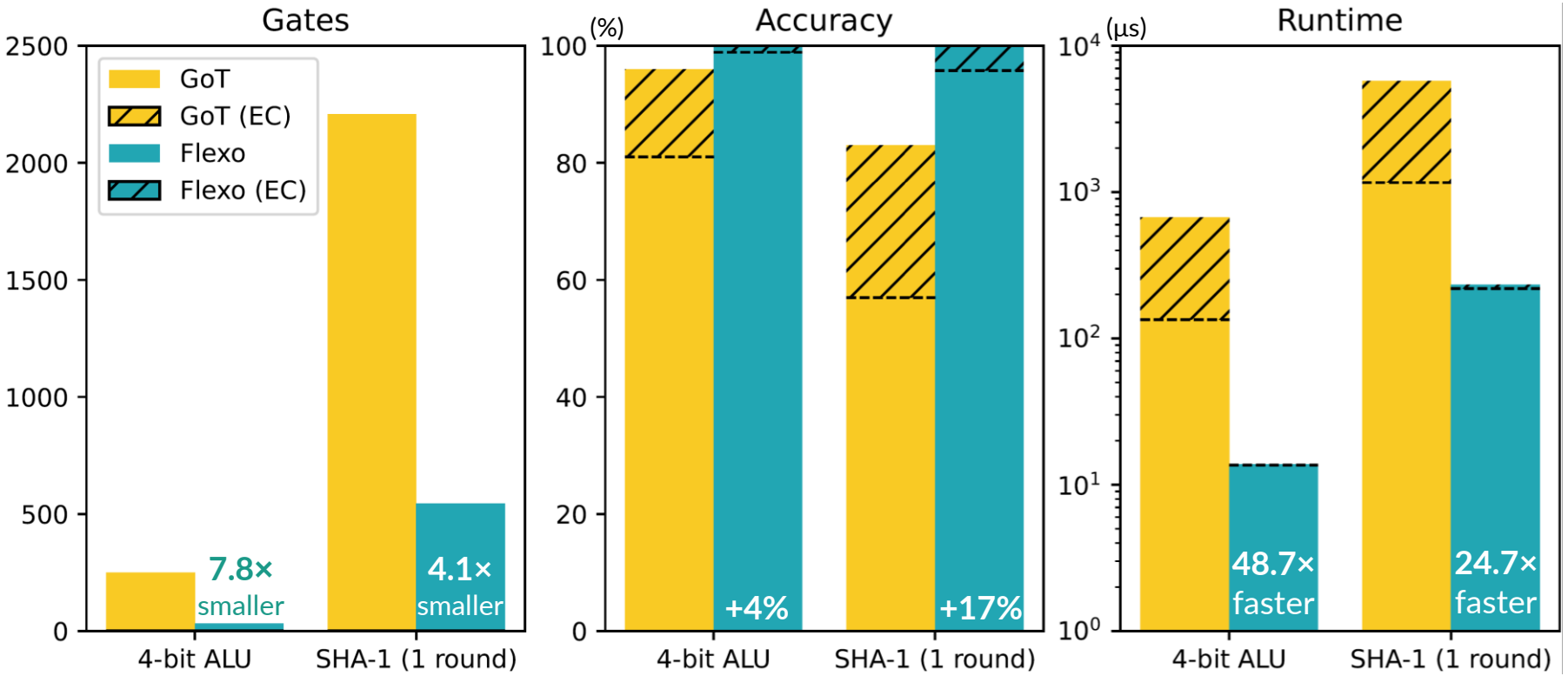
Evaluation: experimental setup

- 8 shared AWS EC2 instances
- All Flexo µWMs are generated by our compiler
- Compare with prior work (GoT) [1] on the Skylake machine

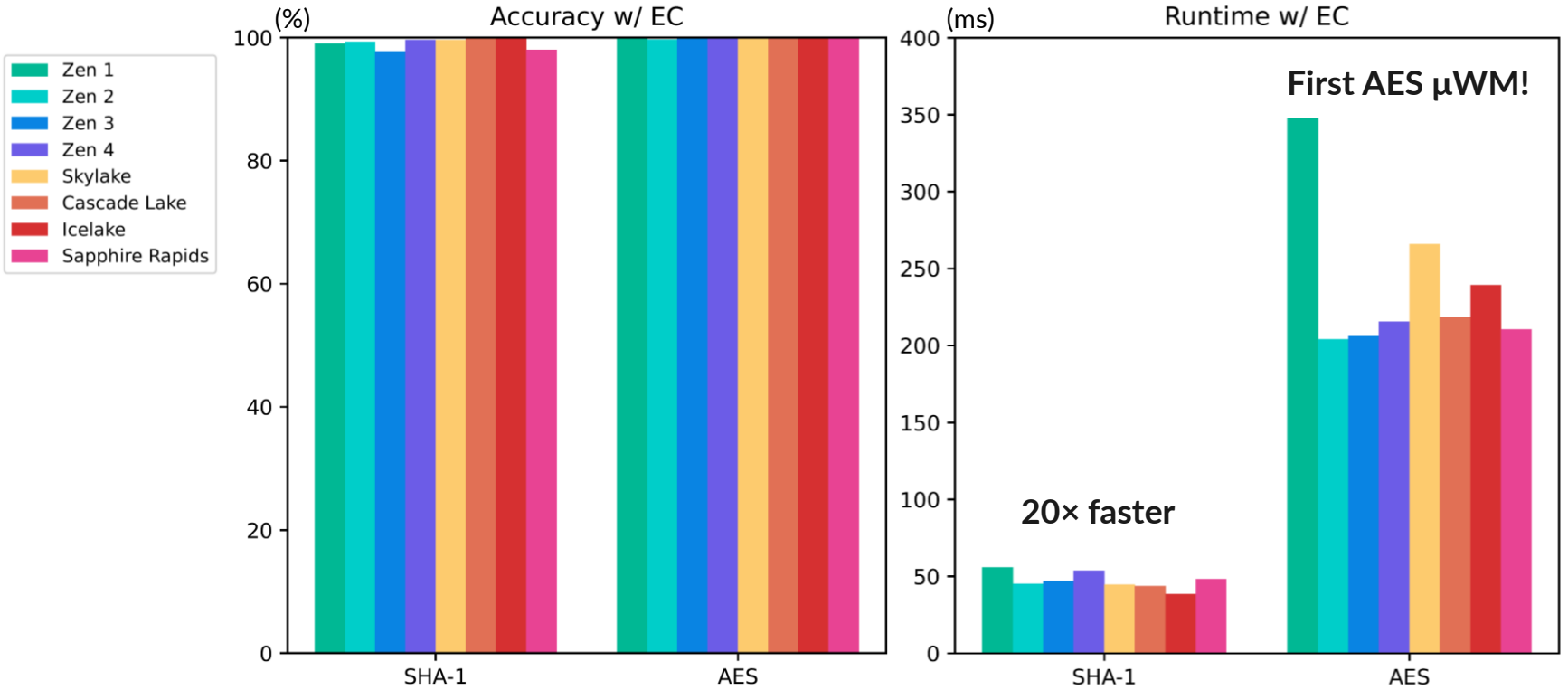
Microarchitecture	Instance type
AMD Zen 1	t3a.xlarge
AMD Zen 2	c5a.xlarge
AMD Zen 3	c6a.xlarge
AMD Zen 4	m7a.xlarge
Intel Skylake	c5n.xlarge
Intel Cascade Lake	m5n.xlarge
Intel Icelake	m6in.xlarge
Intel Sapphire Rapids	m7i.xlarge

[1] The Gates of Time: Improving Cache Attacks with Transient Execution, Katzman et al., USENIX Sec '23

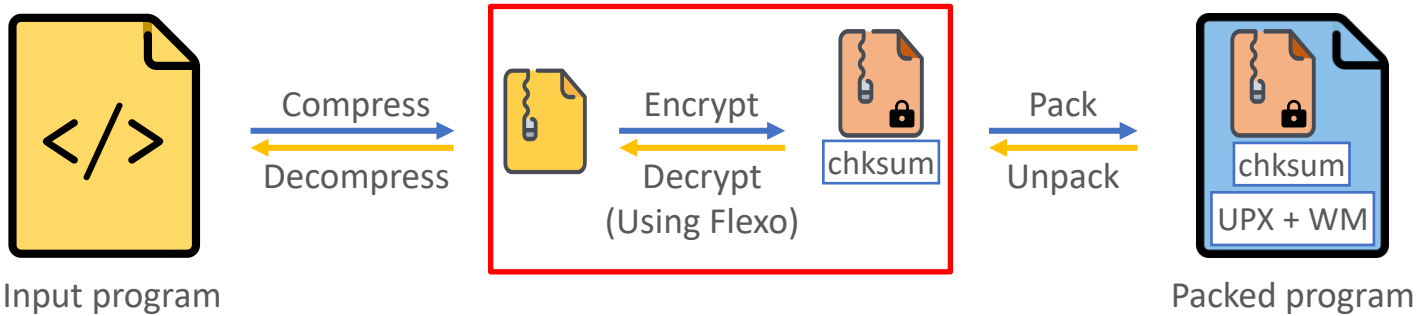
GoT vs. Flexo: 4-bit ALU and SHA-1 (1 round)



Flexo's performance on crypto applications



UPX + Flexo: the first µWM binary packer



Roughly 1 minute to unpack 132 KB using Simon
(5 minutes using AES)

➡ **Slower, but defeats anti-virus tools!**

Takeaways

✨ Flexo makes µWMs much faster and more accurate

✨ Flexo's compiler makes it easy to develop new µWMs

✨ µWMs are practical for program obfuscation or other attacks

▲ GitHub repo

- Bending microarchitectural weird machines towards practicality
- GitHub: https://github.com/joeywang4/Flexo
- Contact: Ping-Lun Wang, pinglunw@andrew.cmu.edu

Carnegie Mellon University
CyLab
Security and Privacy Institute