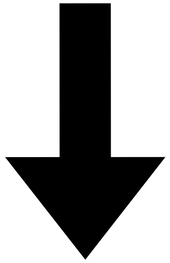
Quantum Advantage for All

Christoph Kirsch, University of Salzburg, Austria and Czech Technical University, Prague, Czechia

Joint work with Daniel Kocher, Stefanie Muroya, and Michael Starzinger

github.com/cksystemsteaching/selfie

1. Self-contained 12-KLOC system
2. Self-compiling C* compiler
3. Self-executing RISC-U emulator
4. Self-hosting RISC-U hypervisor
5. Self-collecting garbage collector
6. Self-fuzzing fuzzer
7. Self-executing symbolic execution engine



github.com/cksystemsgroup/unicorn

RISC-V symbolic execution engine in Rust:

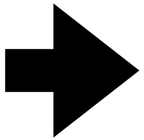
1. Inspired by bounded model checking
2. Connects to Z3, boolector, btormc, SAT solvers
3. Targets quantum annealers and
   gate-model quantum machines
   as accelerators


Dachstein Glacier

Unicorn

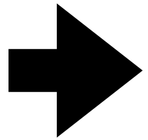
n is the number of instructions on any path

Program P
(64-bit RISC-V)



$O(|P|)$
Finite State Machine
over
SMT of bitvectors/arrays

inspired by [BTOR2, Biere et al.]



$O(n^2)$
Combinational
Circuit

reachability

if and only if

satisfiability

```
uint64_t* x;
uint64_t main() { uint64_t a;
  x = malloc(1); // rounded up to 8
  // touch to trigger page fault here
  *x = 0;
  // read 1 byte from console into x
  read(0, x, 1);
  // copy from heap to stack segment
  a = *x;
  // decrement input until <= '0'
  while (a > '0')
    a = a - 1;
  // segmentation fault on input '1'
  if (a == *x - 1) // '0' == '1' - 1
    // segfault: '0' != 0
    a = *(x + a);
  return 0;
}
```

```
1 sort bitvec 1  ; Boolean
2 sort bitvec 64 ; 64-bit machine word
3 sort array 2 2 ; 64-bit physical memory
10 zero 1 ...
//... register states 200-231 ...
200 zero 2 zero // register $0 is always 0 ...
203 state 2 gp ; register $3 ...
205 state 2 t0 ; register $5
206 state 2 t1 ; register $6
//... program counter states ...
16603600 state 1       // beq t0,zero,8[R0]:
16603601 init 1 16603600 10
16604000 state 1    // A0:ld t0,-16(gp) ...
16606800 state 1    // R0:addi t0,zero,0
//... 64-bit memory (data,heap,stack):
20000000 state 3 physical-memory
  loading data,heap,stack into memory:
20000001 init 3 20000000 17380002
//... data flow ...    A0:ld t0,-16(gp):
36604000 constd 2 -16
36604001 add 2 203 36604000
36604003 read 2 20000000 36604001
36604004 ite 2 16604000 36604003 36603202
//...                  A1:add t0,t0,t1:
36605600 add 2 205 206
36605601 ite 2 16605600 36605600 36604004
//...               SEGFL:ld t0,0(t0):
36606002 ite 2 16606000 36606001 36605601
//...                  R0:addi t0,zero,0:
36606800 ite 2 16606800 200 36606002
//... updating registers ...
60000005 next 2 205 36606800 t0
```

Data Flow

```
uint64_t* x;
uint64_t main() { uint64_t a;
  x = malloc(1); // rounded up to 8
  // touch to trigger page fault here
  *x = 0;
  // read 1 byte from console into x
  read(0, x, 1);
  // copy from heap to stack segment
  a = *x;
  // decrement input until <= '0'
  while (a > '0')
    a = a - 1;
  // segmentation fault on input '1'
  if (a == *x - 1) // '0' == '1' - 1
    // segfault: '0' != 0
    a = *(x + a);
  return 0;
}
```

```
11 one 1
//... data flow ...
36603600 eq 1 205 200    // $t0==$zero
36603601 not 1 36603600 // $t0!=$zero
//... control flow ...
//              beq t0,zero,8[R0]:
56603600 next 1 16603600 16603200
//          A0:ld t0,-16(gp):
56604000 and 1 16603600 36603601
56604001 next 1 16604000 56604000
// ...          sd t0,-8(s0):
56606400 next 1 16606400 16606000
// ...       R0:addi t0,zero,0:
56606800 and 1 16603600 36603600
56606801 ite 1 56606800 11 16606400
56606802 next 1 16606800 56606801
```

Control Flow

```
20 zero 2 ...
22 constd 2 2 ...
//... 1-byte input
71 sort bitvec 8 ; 1 byte ...
81 input 71      ; 1 byte ...
91 uext 2 81 56 // extending input to 64 bits
//... register states ...
202 state 2 sp ; register $2 ...
210 state 2 a0 ; register $10
211 state 2 a1 ; register $11
//... read system call ...
42000001 ite 2 42000000 211 36609200 ...
42000007 eq 1 42000006 22 // inc == 2
42000008 ite 2 42000007 92 91 ...
42000019 eq 1 42000006 28 // inc == 8
42000020 ite 2 42000019 98 42000018
42000021 add 2 211 210 ; $a1 + $a0
//  memory[$a1 + $a0] = input:
42000022 write 3 20000000 42000021 42000020
//... brk system call:
45000001 state 2 brk-bump-pointer
//... updating physical memory:
70000000 next 3 20000000 42000028
//... address >= current end of heap:
80000006 ugte 1 44000001 45000001
// address < current start of stack:
80000007 ult 1 44000001 202
80000008 and 1 80000006 80000007
// access between heap and stack:
80000009 bad 80000008 b2
```

System Calls, Bad States

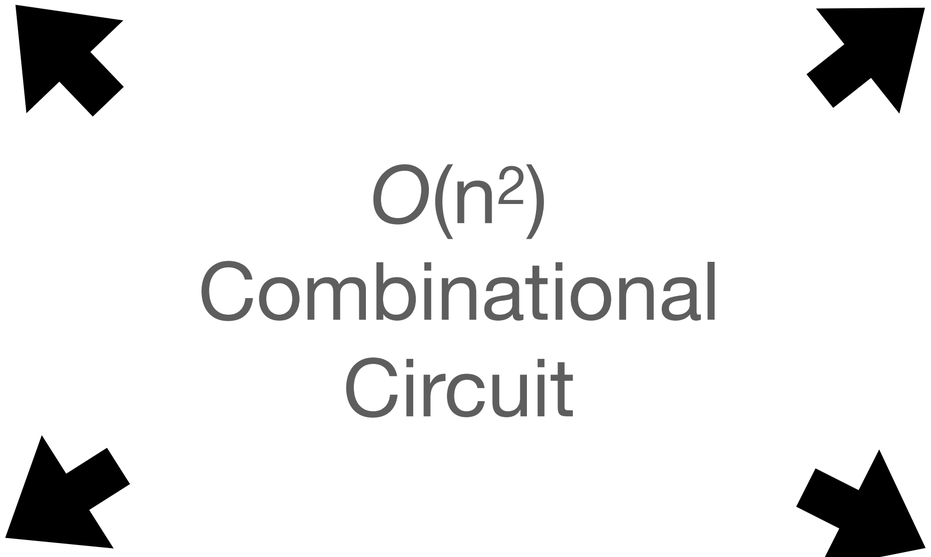
Classic vs Quantum

n is the number of instructions on any path, w is word size

$O(n^2)$
SMT Formula

$O(n^2w^2)$
Quadratic
Unconstrained
Binary Optimization
(QUBO) Model

“4xy - 2x - 2y + 2 = 0”



classical computing

$O(n^2)$
Combinational
Circuit

quantum computing

$O(n^2w^2)$
SAT Formula

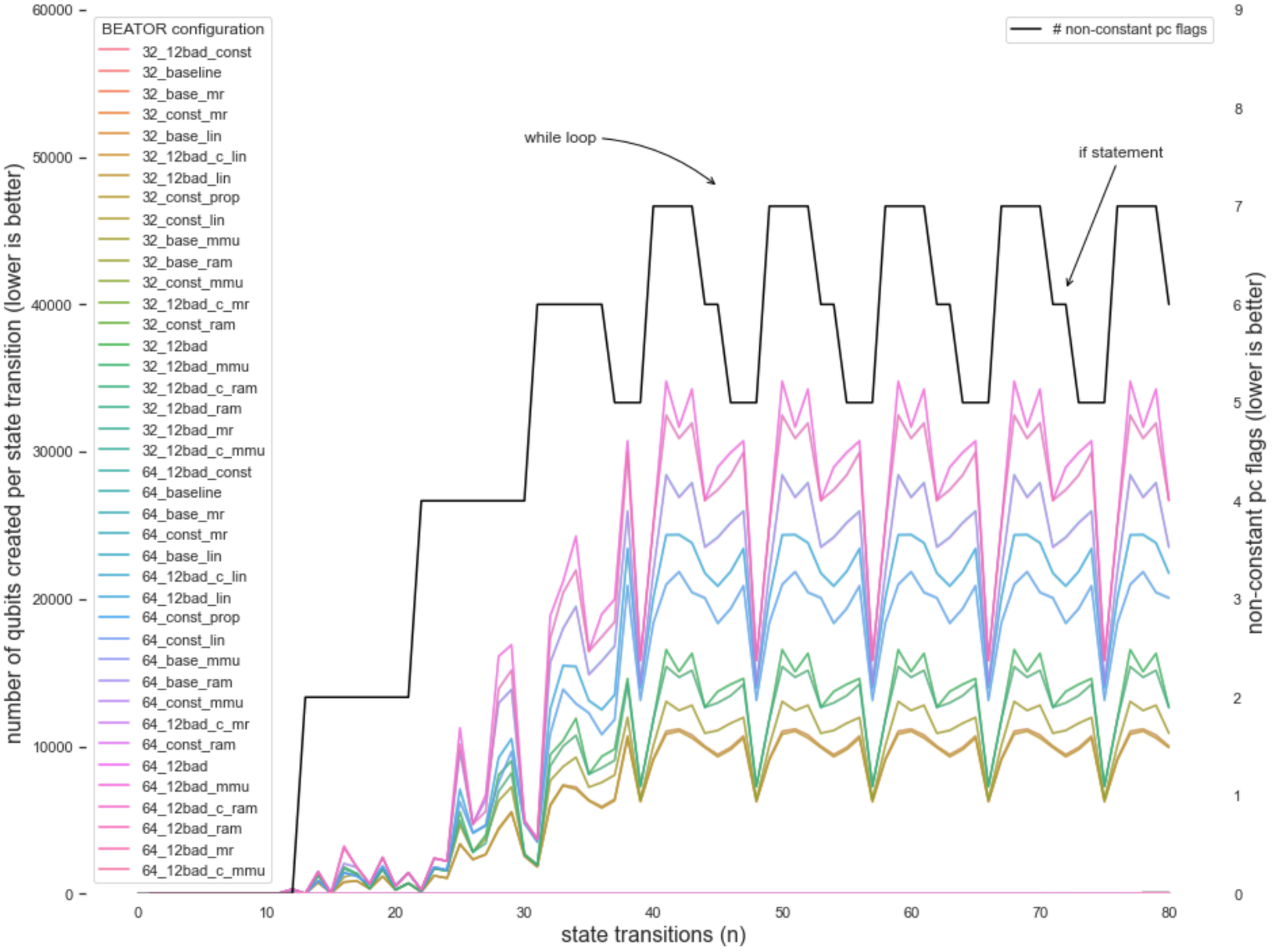
$O(n^2w)$
Quantum Circuit

Algorithmic Time is Quantum Space

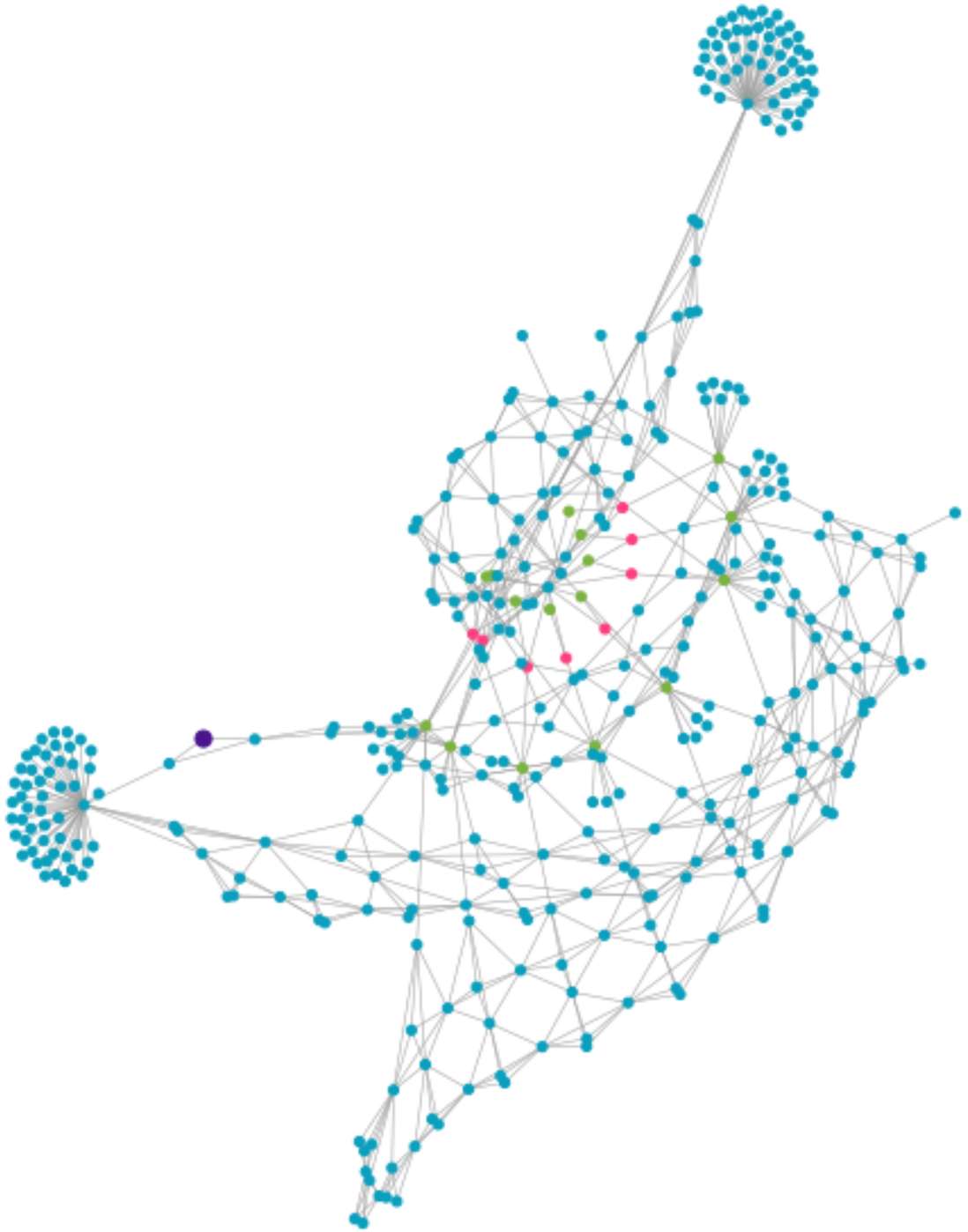
$O(f(x))$ time, $O(g(x))$ space => $O(f(x) \cdot g(x))$ quantum space

$O(f(x))$ to $O(f^2(x))$ quantum space


Winter Rose


BEATOR configuration
32_12bad_const
32_baseline
32_base_mr
32_const_mr
32_base_lin
32_12bad_c_lin
32_12bad_lin
32_const_prop
32_const_lin
32_base_mmu
32_base_ram
32_const_mmu
32_12bad_c_mr
32_const_ram
32_12bad
32_12bad_mmu
32_12bad_c_ram
32_12bad_ram
32_12bad_mr
32_12bad_c_mmu
64_12bad_const
64_baseline
64_base_mr
64_const_mr
64_base_lin
64_12bad_c_lin
64_12bad_lin
64_const_prop
64_const_lin
64_base_mmu
64_base_ram
64_const_mmu
64_12bad_c_mr
64_const_ram
64_12bad
64_12bad_mmu
64_12bad_c_ram
64_12bad_ram
64_12bad_mr
64_12bad_c_mmu
non-constant pc flags
while loop
if statement
number of qubits created per state transition (lower is better)
non-constant pc flags (lower is better)
state transitions (n)
0
10000
20000
30000
40000
50000
60000
0
1
2
3
4
5
6
7
8
9
0
10
20
30
40
50
60
70
80

QUBO



Chimera
Minor Embedding
on
D-Wave
Quantum Annealer

Quantum Advantage for All

https://arxiv.org/abs/2111.12063