SLOTHY: Using Constraint-Solving for Superoptimization of Cryptographic Assembly

Tutorial (with assignment solutions)

Amin Abdulrahman, Max Planck Institute for Security and Privacy, Germany
Matthias J. Kannwischer, Chelpis Quantum Corp., Taiwan

September 14, 2025

Tutorial Structure

Slides & Assignments

Tutorial Assignments

github.com/dop-amin/ches2025-slothy-tutorial

Tutorial Slides

kannwischer.eu/talks/20250914_slothy.pdf

Motivation

Cryptographic Engineering is (Not) Easy

”Hey, we need a fast ML-KEM implementation for our new smartphone CPU. Can you do that?”

Cryptographic Engineering is (Not) Easy

”Hey, we need a fast ML-KEM implementation for our new smartphone CPU. Can you do that?”

”Should be easy, right? I’ll just use the C reference and take it from there”

Cryptographic Engineering is (Not) Easy

”Hey, we need a fast ML-KEM implementation for our new smartphone CPU. Can you do that?”

”Should be easy, right? I’ll just use the C reference and take it from there”

Narrator: It was not easy.

Why Cryptographic Engineering is (Not) Easy

Why Cryptographic Engineering is (Not) Easy

- Simplicity

```
void ntt(int16_t r[256]) {
  unsigned int len, start, j, k;
  int16_t t, zeta;

  k = 1;
  for(len = 128; len >= 2; len >>= 1) {
    for(start = 0; start < 256; start = j + len) {
      zeta = zetas[k++];
      for(j = start; j < start + len; j++) {
        t = fqmul(zeta, r[j + len]);
        r[j + len] = r[j] - t;
        r[j] = r[j] + t;
      }
    }
  }
}
```

Why Cryptographic Engineering is (Not) Easy

- Simplicity
- Security

```
for (i = 0; i < KYBER_N / 8; i++){
  for (j = 0; j < 8; j++){
    mask = -(int16_t)((msg[i] >> j) & 1);
    r->coeffs[8 * i + j] = mask & ((KYBER_Q + 1) / 2);
  }
}
```

Why Cryptographic Engineering is (Not) Easy

- Simplicity
- Security
- Performance

```
.macro mulmodq dst, src, const, idx0, idx1
  vqrdmulhq   t2,  \src, \const, \idx1
  vmulq       \dst,  \src, \const, \idx0
  vmlaq       \dst,  t2, consts, 0
.endm
```

Why Cryptographic Engineering is (Not) Easy

- Simplicity
- Security
- Performance
- More Performance

```
mul v8.8H, v8.8H, v1.H[4]
add v3.8H, v3.8H, v9.8H
sub v9.8H, v12.8H, v10.8H
add v12.8H, v12.8H, v10.8H
mls v8.8H, v16.8H, v7.H[0]
str q3, [x0], #(16)
ldr q10, [x0, #0]
sub v3.8H, v18.8H, v8.8H
add v16.8H, v18.8H, v8.8H
str q23, [x0, #48]
ldr q5, [x0, #64]
str q12, [x0, #112]
```

Why Cryptographic Engineering is (Not) Easy

- Simplicity
- Security
- Performance
- More Performance
- Effort

```
add v18.8H, v8.8H, v22.8H
mul v27.8H, v31.8H, v0.H[0]
add v22.8H, v16.8H, v11.8H
sqrdmulh v14.8H, v31.8H, v0.H[1]
str q4, [x0, #304]
str q18, [x0, #240]
sub v4.8H, v10.8H, v5.8H
add v18.8H, v10.8H, v5.8H
sqrdmulh v24.8H, v22.8H, v0.H[3]
mul v10.8H, v22.8H, v0.H[2]
str q4, [x0, #432]
str q18, [x0, #368]
```

Why Cryptographic Engineering is (Not) Easy

- Simplicity
- Security
- Performance
- More Performance
- Effort
- Auditability & Maintainability

```
mul v8.8H, v8.8H, v1.H[4]
add v3.8H, v3.8H, v9.8H
sub v9.8H, v12.8H, v10.8H
add v12.8H, v12.8H, v10.8H
mls v8.8H, v16.8H, v7.H[0]
str q3, [x0], #(16)
ldr q10, [x0, #0]
sub v3.8H, v18.8H, v8.8H
add v16.8H, v18.8H, v8.8H
str q23, [x0, #48]
ldr q5, [x0, #64]
str q12, [x0, #112]
```

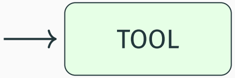
```
add v18.8H, v8.8H, v22.8H
mul v27.8H, v31.8H, v0.H[0]
add v22.8H, v16.8H, v11.8H
sqrdmulh v14.8H, v31.8H, v0.H[1]
str q4, [x0, #304]
str q18, [x0, #240]
sub v4.8H, v10.8H, v5.8H
add v18.8H, v10.8H, v5.8H
sqrdmulh v24.8H, v22.8H, v0.H[3]
mul v10.8H, v22.8H, v0.H[2]
str q4, [x0, #432]
str q18, [x0, #368]
```

SLOTHY High-Level Workflow

```
.macro mulmodq dst, src, const, idx0, idx1
  sqrdmulh tmp2.8h, \src.8h, \const.h[\idx1]
  mul \dst.8h, \src.8h, \const.h[\idx0]
  mla \dst.8h, tmp2.8h, consts.h[0]
.endm
.macro ct_butterfly a, b, root, idx0, idx1
  mulmodq tmp, \b, \root, \idx0, \idx1
  sub \b.8h, \a.8h, tmp.8h
  add \a.8h, \a.8h, tmp.8h
.endm

ct_butterfly data0, data8, root0, 0, 1
ct_butterfly data1, data9, root0, 0, 1
ct_butterfly data2, data10, root0, 0, 1
ct_butterfly data3, data11, root0, 0, 1
ct_butterfly data4, data12, root0, 0, 1
ct_butterfly data5, data13, root0, 0, 1
ct_butterfly data6, data14, root0, 0, 1
ct_butterfly data7, data15, root0, 0, 1
```

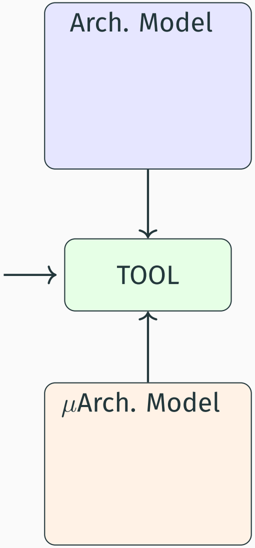
SLOTHY High-Level Workflow

```
.macro mulmodq dst, src, const, idx0, idx1
  sqrdmulh tmp2.8h, \src.8h, \const.h[\idx1]
  mul \dst.8h, \src.8h, \const.h[\idx0]
  mla \dst.8h, tmp2.8h, consts.h[0]
.endm
.macro ct_butterfly a, b, root, idx0, idx1
  mulmodq tmp, \b, \root, \idx0, \idx1
  sub \b.8h, \a.8h, tmp.8h
  add \a.8h, \a.8h, tmp.8h
.endm

ct_butterfly data0, data8, root0, 0, 1
ct_butterfly data1, data9, root0, 0, 1
ct_butterfly data2, data10, root0, 0, 1
ct_butterfly data3, data11, root0, 0, 1
ct_butterfly data4, data12, root0, 0, 1
ct_butterfly data5, data13, root0, 0, 1
ct_butterfly data6, data14, root0, 0, 1
ct_butterfly data7, data15, root0, 0, 1
```



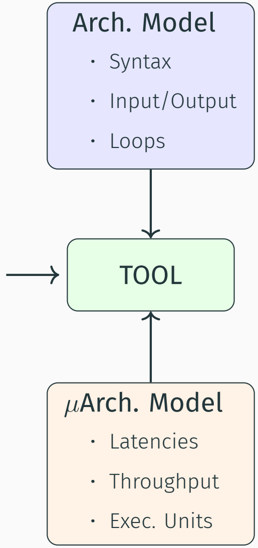
SLOTHY High-Level Workflow

```
.macro mulmodq dst, src, const, idx0, idx1
  sqrdmulh tmp2.8h, \src.8h, \const.h[\idx1]
  mul \dst.8h, \src.8h, \const.h[\idx0]
  mla \dst.8h, tmp2.8h, consts.h[0]
.endm
.macro ct_butterfly a, b, root, idx0, idx1
  mulmodq tmp, \b, \root, \idx0, \idx1
  sub \b.8h, \a.8h, tmp.8h
  add \a.8h, \a.8h, tmp.8h
.endm

ct_butterfly data0, data8, root0, 0, 1
ct_butterfly data1, data9, root0, 0, 1
ct_butterfly data2, data10, root0, 0, 1
ct_butterfly data3, data11, root0, 0, 1
ct_butterfly data4, data12, root0, 0, 1
ct_butterfly data5, data13, root0, 0, 1
ct_butterfly data6, data14, root0, 0, 1
ct_butterfly data7, data15, root0, 0, 1
```

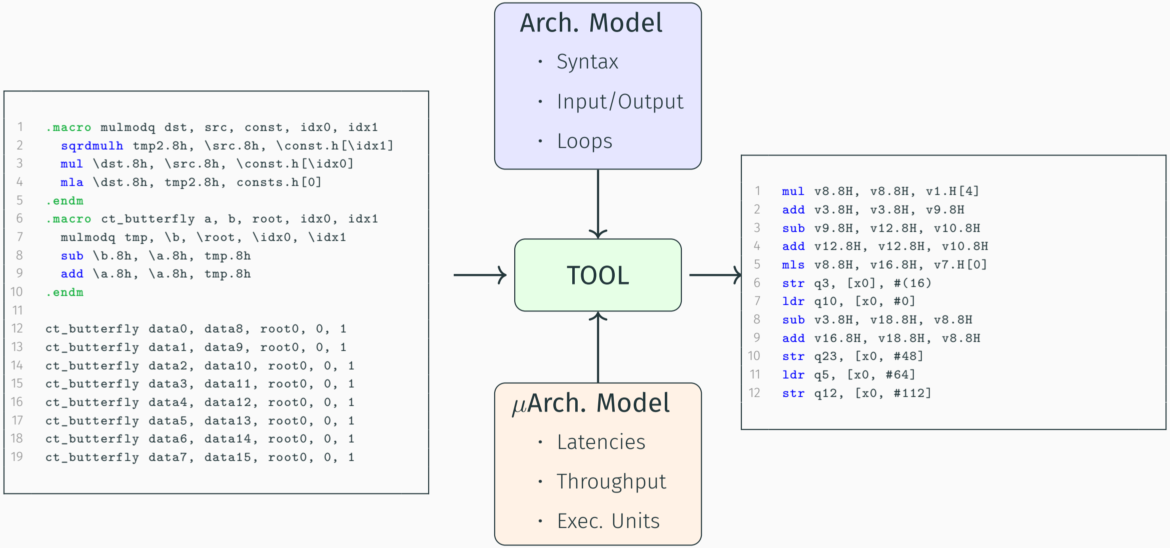


SLOTHY High-Level Workflow

```
.macro mulmodq dst, src, const, idx0, idx1
  sqrdmulh tmp2.8h, \src.8h, \const.h[\idx1]
  mul \dst.8h, \src.8h, \const.h[\idx0]
  mla \dst.8h, tmp2.8h, consts.h[0]
.endm
.macro ct_butterfly a, b, root, idx0, idx1
  mulmodq tmp, \b, \root, \idx0, \idx1
  sub \b.8h, \a.8h, tmp.8h
  add \a.8h, \a.8h, tmp.8h
.endm

ct_butterfly data0, data8, root0, 0, 1
ct_butterfly data1, data9, root0, 0, 1
ct_butterfly data2, data10, root0, 0, 1
ct_butterfly data3, data11, root0, 0, 1
ct_butterfly data4, data12, root0, 0, 1
ct_butterfly data5, data13, root0, 0, 1
ct_butterfly data6, data14, root0, 0, 1
ct_butterfly data7, data15, root0, 0, 1
```



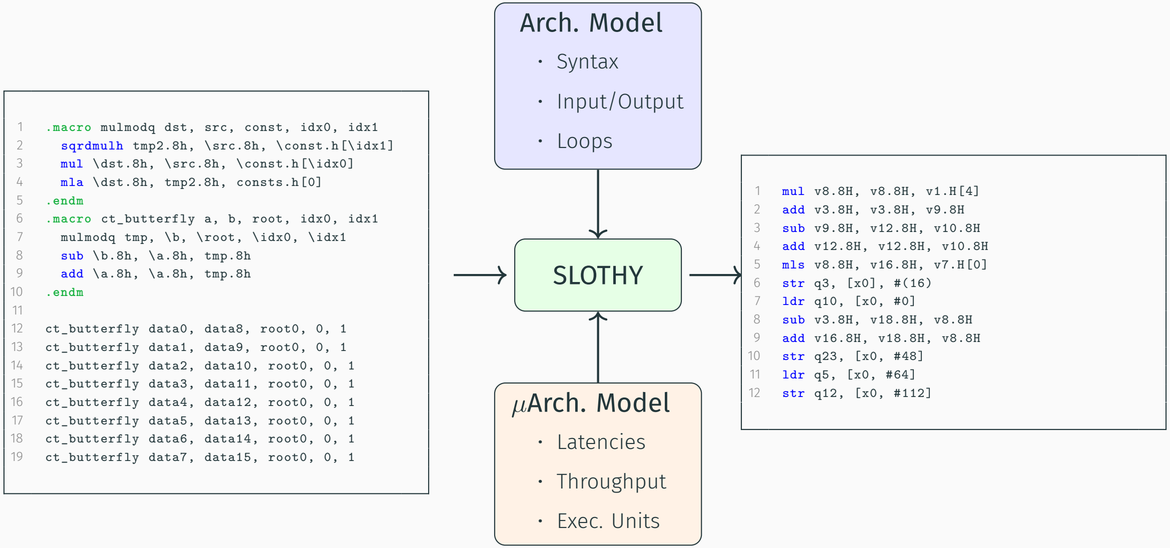
SLOTHY High-Level Workflow



```
.macro mulmodq dst, src, const, idx0, idx1
  sqrdmulh tmp2.8h, \src.8h, \const.h[\idx1]
  mul \dst.8h, \src.8h, \const.h[\idx0]
  mla \dst.8h, tmp2.8h, consts.h[0]
.endm
.macro ct_butterfly a, b, root, idx0, idx1
  mulmodq tmp, \b, \root, \idx0, \idx1
  sub \b.8h, \a.8h, tmp.8h
  add \a.8h, \a.8h, tmp.8h
.endm

ct_butterfly data0, data8, root0, 0, 1
ct_butterfly data1, data9, root0, 0, 1
ct_butterfly data2, data10, root0, 0, 1
ct_butterfly data3, data11, root0, 0, 1
ct_butterfly data4, data12, root0, 0, 1
ct_butterfly data5, data13, root0, 0, 1
ct_butterfly data6, data14, root0, 0, 1
ct_butterfly data7, data15, root0, 0, 1
```

```
mul v8.8H, v8.8H, v1.H[4]
add v3.8H, v3.8H, v9.8H
sub v9.8H, v12.8H, v10.8H
add v12.8H, v12.8H, v10.8H
mls v8.8H, v16.8H, v7.H[0]
str q3, [x0], #(16)
ldr q10, [x0, #0]
sub v3.8H, v18.8H, v8.8H
add v16.8H, v18.8H, v8.8H
str q23, [x0, #48]
ldr q5, [x0, #64]
str q12, [x0, #112]
```

SLOTHY High-Level Workflow



```
.macro mulmodq dst, src, const, idx0, idx1
  sqrdmulh tmp2.8h, \src.8h, \const.h[\idx1]
  mul \dst.8h, \src.8h, \const.h[\idx0]
  mla \dst.8h, tmp2.8h, consts.h[0]
.endm
.macro ct_butterfly a, b, root, idx0, idx1
  mulmodq tmp, \b, \root, \idx0, \idx1
  sub \b.8h, \a.8h, tmp.8h
  add \a.8h, \a.8h, tmp.8h
.endm

ct_butterfly data0, data8, root0, 0, 1
ct_butterfly data1, data9, root0, 0, 1
ct_butterfly data2, data10, root0, 0, 1
ct_butterfly data3, data11, root0, 0, 1
ct_butterfly data4, data12, root0, 0, 1
ct_butterfly data5, data13, root0, 0, 1
ct_butterfly data6, data14, root0, 0, 1
ct_butterfly data7, data15, root0, 0, 1
```

```
mul v8.8H, v8.8H, v1.H[4]
add v3.8H, v3.8H, v9.8H
sub v9.8H, v12.8H, v10.8H
add v12.8H, v12.8H, v10.8H
mls v8.8H, v16.8H, v7.H[0]
str q3, [x0], #(16)
ldr q10, [x0, #0]
sub v3.8H, v18.8H, v8.8H
add v16.8H, v18.8H, v8.8H
str q23, [x0, #48]
ldr q5, [x0, #64]
str q12, [x0, #112]
```

What is SLOTHY?

SLOTHY: Super (Lazy) Optimization of Tricky Handwritten assemblY

What is SLOTHY?

SLOTHY: Super (Lazy) Optimization of Tricky Handwritten assemblY

A **fixed-instruction superoptimizer** that:

- Takes your assembly code as input
- Preserves your instruction choices (security!)
- Simultaneously solves
    - instruction scheduling
    - register allocation
    - software pipelining

What is SLOTHY?

SLOTHY: Super (Lazy) Optimization of Tricky Handwritten assemblY

A **fixed-instruction superoptimizer** that:

- Takes your assembly code as input
- Preserves your instruction choices (security!)
- Simultaneously solves
  - instruction scheduling
  - register allocation
  - software pipelining

Finding Optimal Solutions

Formulates optimization as a **constraint satisfaction problem** and uses Google OR-Tools' CP-SAT to find optimal solutions.

Supported Platforms

Partial models for a variety of architectures and microarchitectures:

AArch64 (+ Neon)

- Arm Cortex-A55
- Arm Cortex-A72
- Arm Neoverse N1
- Apple M1

Armv8-M (+ MVE)

- Arm Cortex-M55
- Arm Cortex-M85

Armv7E-M

- Arm Cortex-M7
- Arm Cortex-M4 (WIP)

RISC-V (WIP)

- XuanTie C908 (WIP)
- SpacemiT X60 (WIP)

SLOTHY: Does it do any good?

Workload	μArch	Before (cycles)	After (cycles)	Speed-up
ML-DSA NTT	Cortex-A55	2436	1728	1.41×
ML-DSA NTT	Cortex-A72	2241	1766	1.27×
ML-DSA NTT	Cortex-M7	8139	4141	1.97×
X25519	Cortex-A55	143849	139752	1.03×
Keccak-f1600	Cortex-M7	6691	5149	1.30×

Also successfully applied to complex FFT, more subroutines from ML-DSA & ML-KEM

A. Abdulrahman, H. Becker, M. J. Kannwischer, and F. Klein. *Fast and Clean: Auditable high-performance assembly via constraint solving.* CHES '24. 2023

A. Abdulrahman, M. J. Kannwischer, and T.-H. Lim. "Enabling Microarchitectural Agility: Taking ML-KEM & ML-DSA from Cortex-M4 to M7 with SLOTHY". In: ASIA CCS '25. 2025

SLOTHY is Used in Practice

SLOTHY-optimized code is used in practice:

- **AWS libcrypto (AWS-LC)**: SLOTHY-optimized X/Ed25519, P256, P384, P521, Keccak, and ML-KEM code has been merged into AWS-LC as part of s2n-bignum.

[1] D. Kostic, H. Becker, J. Harrison, J. Lee, N. Ebeid, and T. Hansen. ***Adoption of High-Assurance and Highly Performant Cryptographic Algorithms at AWS.*** Real World Crypto 2024. Amazon Web Services, Apr. 2024

SLOTHY is Used in Practice

SLOTHY-optimized code is used in practice:

- **AWS libcrypto (AWS-LC)**: SLOTHY-optimized X/Ed25519, P256, P384, P521, Keccak, and ML-KEM code has been merged into AWS-LC as part of s2n-bignum.
  $\rightarrow$ *"Servicing Trillions of requests a day"*[1]

[1] D. Kostic, H. Becker, J. Harrison, J. Lee, N. Ebeid, and T. Hansen. ***Adoption of High-Assurance and Highly Performant Cryptographic Algorithms at AWS.*** Real World Crypto 2024. Amazon Web Services, Apr. 2024

SLOTHY is Used in Practice

SLOTHY-optimized code is used in practice:

- **AWS libcrypto (AWS-LC)**: SLOTHY-optimized X/Ed25519, P256, P384, P521, Keccak, and ML-KEM code has been merged into AWS-LC as part of s2n-bignum.
  $\rightarrow$ *"Servicing Trillions of requests a day"*[1]
- **s2n-bignum**: AWS' bignum library routinely employs SLOTHY for finding further highly optimized ECC implementations.

[1] D. Kostic, H. Becker, J. Harrison, J. Lee, N. Ebeid, and T. Hansen. ***Adoption of High-Assurance and Highly Performant Cryptographic Algorithms at AWS.*** Real World Crypto 2024. Amazon Web Services, Apr. 2024

SLOTHY is Used in Practice

SLOTHY-optimized code is used in practice:

- **AWS libcrypto (AWS-LC)**: SLOTHY-optimized X/Ed25519, P256, P384, P521, Keccak, and ML-KEM code has been merged into AWS-LC as part of s2n-bignum.
  → *"Servicing Trillions of requests a day"*[1]
- **s2n-bignum**: AWS' bignum library routinely employs SLOTHY for finding further highly optimized ECC implementations.
- **Arm EndpointAI**: SLOTHY-optimized code has been deployed to the CMSIS DSP Library for the radix-4 CFFT routines as part of the Arm EndpointAI project

[1]D. Kostic, H. Becker, J. Harrison, J. Lee, N. Ebeid, and T. Hansen. ***Adoption of High-Assurance and Highly Performant Cryptographic Algorithms at AWS.*** Real World Crypto 2024. Amazon Web Services, Apr. 2024

SLOTHY is Used in Practice

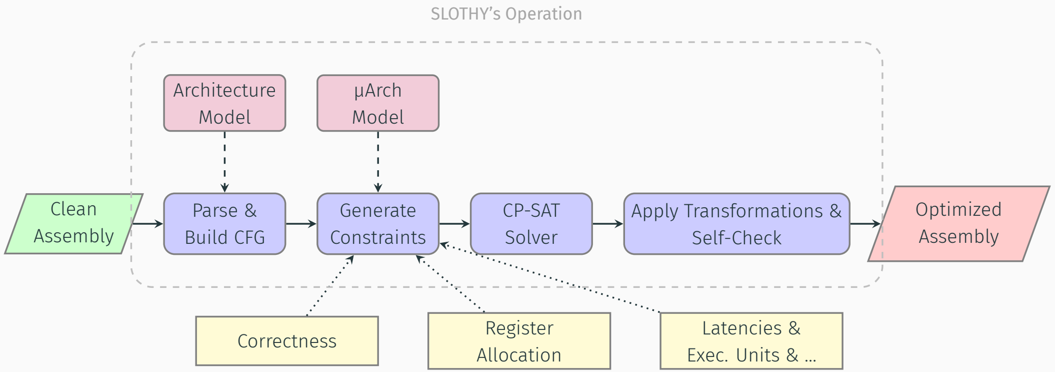
SLOTHY-optimized code is used in practice:

- **AWS libcrypto (AWS-LC)**: SLOTHY-optimized X/Ed25519, P256, P384, P521, Keccak, and ML-KEM code has been merged into AWS-LC as part of s2n-bignum.
  $\rightarrow$ *"Servicing Trillions of requests a day"*[1]
- **s2n-bignum**: AWS' bignum library routinely employs SLOTHY for finding further highly optimized ECC implementations.
- **Arm EndpointAI**: SLOTHY-optimized code has been deployed to the CMSIS DSP Library for the radix-4 CFFT routines as part of the Arm EndpointAI project
- **mlkem-native**: AArch64 assembly routines of ML-KEM are automatically optimized using SLOTHY. **See Matthias talk at OPTIMIST workshop today at 2pm**

[1]D. Kostic, H. Becker, J. Harrison, J. Lee, N. Ebeid, and T. Hansen. ***Adoption of High-Assurance and Highly Performant Cryptographic Algorithms at AWS.*** Real World Crypto 2024. Amazon Web Services, Apr. 2024

Inside SLOTHY

Inside SLOTHY: Workflow

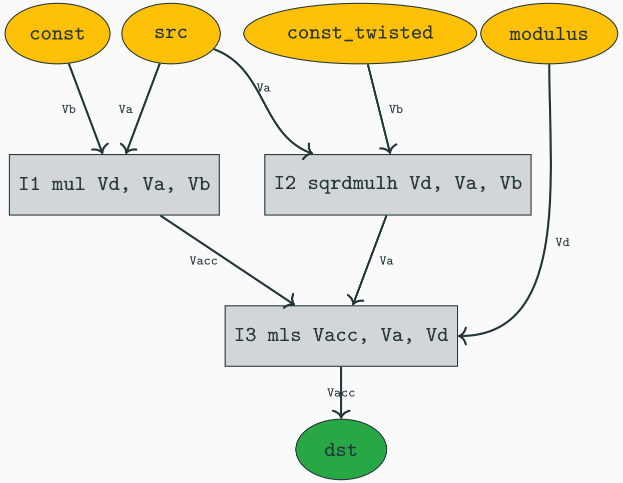


Inside SLOTHY: Parsing the Code

Input Assembly

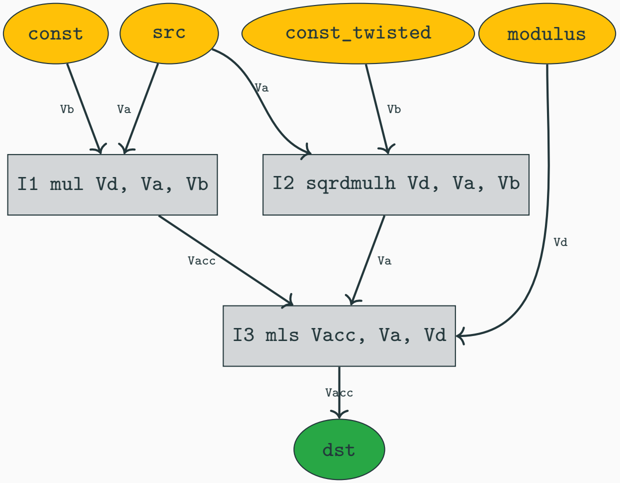
```
// in: src, modulus, const,
       const_twisted
mul      dst, src, const
sqrdmulh tmp, src, const_twisted
mls      dst, tmp, modulus
// out: dst

```

Computational Flow Graph



Inside SLOTHY: Construction of Constraints, Correctness

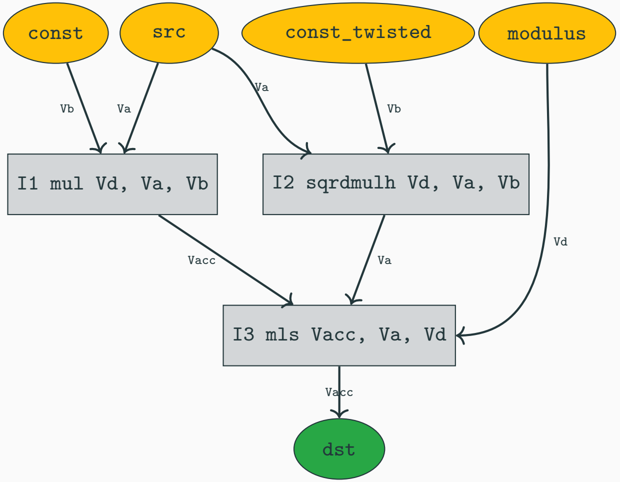
Computational Flow Graph



- Each instruction has a program position `IX.pos` (integer variable)
- Naturally, each position can only be assigned once
- For correctness, consumer after producer:
  - `I1.pos` $<$ `I3.pos`
  - `I2.pos` $<$ `I3.pos`

Inside SLOTHY: Construction of Constraints, Register Allocation

Computational Flow Graph



Boolean variables (reg. is output)

`I1.V0, ..., I1.V31`   `I2.V0, ..., I2.V31`

`I3` needs the same output register as `I1`

Reg. alloc. constraint

Exactly one of `I1.V0, …, I1.V31` is true

Reg. usage interval (cond. on boolean variables)

`[I1.pos, I3.pos]`   `[I2.pos, I3.pos]`

Lifetime constraint

For each register: Usage intervals must not overlap

Inside SLOTHY: What Else are we Modeling?

- Performance characteristics:
    - Latencies: $\texttt{I1.pos} + 3 < \texttt{I3.pos}$

Inside SLOTHY: What Else are we Modeling?

- Performance characteristics:
  - Latencies: $\texttt{I1.pos} + 3 < \texttt{I3.pos}$
  - Occupancy of execution units
  - Throughput, i.e., how long is a unit kept busy?

Inside SLOTHY: What Else are we Modeling?

- Performance characteristics:
  - Latencies: $\texttt{I1.pos} + 3 < \texttt{I3.pos}$
  - Occupancy of execution units
  - Throughput, i.e., how long is a unit kept busy?
  - Forwarding paths

Inside SLOTHY: What Else are we Modeling?

- Performance characteristics:
  - Latencies: $\texttt{I1.pos} + 3 < \texttt{I3.pos}$
  - Occupancy of execution units
  - Throughput, i.e., how long is a unit kept busy?
  - Forwarding paths
  - Stalls: In case there is no perfect solution, we model “gaps” in the scheduling
    - Binary search: Find solution for a fixed number of stalls; performs (external) binary search to find minimum
    - Variable size: Model number of stalls within the constraint model (recommended for small to medium-sized examples)

Inside SLOTHY: What Else are we Modeling?

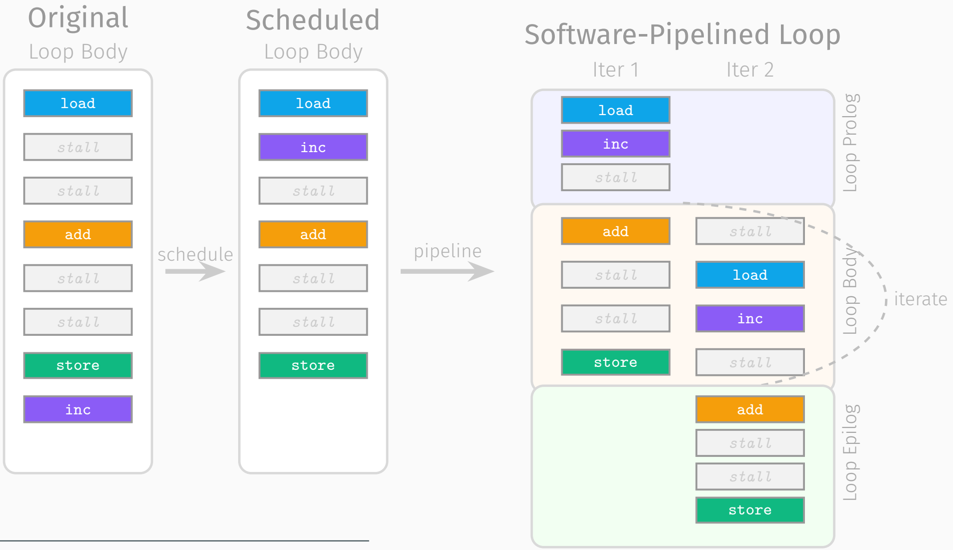
- Performance characteristics:
  - Latencies: $\texttt{I1.pos} + 3 < \texttt{I3.pos}$
  - Occupancy of execution units
  - Throughput, i.e., how long is a unit kept busy?
  - Forwarding paths
  - Stalls: In case there is no perfect solution, we model “gaps” in the scheduling
    - **Binary search:** Find solution for a fixed number of stalls; performs (external) binary search to find minimum
    - **Variable size:** Model number of stalls within the constraint model (recommended for small to medium-sized examples)
- Memory dependencies
- Stack spills

Inside SLOTHY: What Else are we Modeling?

- Performance characteristics:
    - Latencies: $\texttt{I1.pos} + 3 < \texttt{I3.pos}$
    - Occupancy of execution units
    - Throughput, i.e., how long is a unit kept busy?
    - Forwarding paths
    - Stalls: In case there is no perfect solution, we model “gaps” in the scheduling
        - **Binary search:** Find solution for a fixed number of stalls; performs (external) binary search to find minimum
        - **Variable size:** Model number of stalls within the constraint model (recommended for small to medium-sized examples)
- Memory dependencies
- Stack spills

Note on scalability: The more properties we model and the more instructions we have, the more complex the constraint problem might be.
$\Longrightarrow$ At some point, problem becomes infeasible to solve
$\Longrightarrow$ Workaround: Splitting heuristic optimizes the code piece by piece

Inside SLOTHY: Software Pipelining



Visualization inspired by `https://www.lighterra.com/papers/basicinstructionscheduling/`.

Symbolic Registers in SLOTHY

Symbolic Registers

- Manual register allocation can be tedious
- SLOTHY will re-allocate registers anyway
- We might as well leave the entire register allocation to SLOTHY

Symbolic Registers in SLOTHY

Symbolic Registers

- Manual register allocation can be tedious
- SLOTHY will re-allocate registers anyway
- We might as well leave the entire register allocation to SLOTHY

Traditional Assembly

```
// Manual register allocation
ldr q0, [x1], #16
ldr q1, [x2], #16
add v0.8h, v0.8h, v1.8h
str q0, [x0], #16
```

With Symbolic Registers

```
// SLOTHY allocates registers
ldr q<a>, [x1], #16
ldr q<b>, [x2], #16
add v<c>.8h, v<a>.8h, v<b>.8h
str q<c>, [x0], #16
```

Symbolic Registers in SLOTHY

Symbolic Registers

- Manual register allocation can be tedious
- SLOTHY will re-allocate registers anyway
- We might as well leave the entire register allocation to SLOTHY

Traditional Assembly

```
// Manual register allocation
ldr q0, [x1], #16
ldr q1, [x2], #16
add v0.8h, v0.8h, v1.8h
str q0, [x0], #16
```

With Symbolic Registers

```
// SLOTHY allocates registers
ldr q<a>, [x1], #16
ldr q<b>, [x2], #16
add v<c>.8h, v<a>.8h, v<b>.8h
str q<c>, [x0], #16
```

Syntax: `type<name>`

- Example: `x<A>` (64-bit), `w<A>` (32-bit view of same register)
- The constraint problem complexity is unchanged (as renaming is done anyway)

Symbolic Registers in SLOTHY

Why Type Prefix is Necessary

- Problem 1: `add A, B, C` - scalar or vector operation?
- Problem 2: Arm has register views, e.g., `x0-x30` (64-bit), `w0-w30` (32-bit lower half)
  $\implies$ Need to be able to tell apart `add x0, x1, x2` and `add w0, w1, w2`

Symbolic Registers in SLOTHY

Why Type Prefix is Necessary

- Problem 1: `add A, B, C` - scalar or vector operation?
- Problem 2: Arm has register views, e.g., `x0-x30` (64-bit), `w0-w30` (32-bit lower half)
  $\Longrightarrow$ Need to be able to tell apart `add x0, x1, x2` and `add w0, w1, w2`

Benefits of using symbolic registers

- Developer does not have to think about register allocation
- Can write code with no valid register allocation without reordering

Symbolic Registers in SLOTHY

Why Type Prefix is Necessary

- Problem 1: `add A, B, C` - scalar or vector operation?
- Problem 2: Arm has register views, e.g., `x0-x30` (64-bit), `w0-w30` (32-bit lower half)
  $\Longrightarrow$ Need to be able to tell apart `add x0, x1, x2` and `add w0, w1, w2`

Benefits of using symbolic registers

- Developer does not have to think about register allocation
- Can write code with no valid register allocation without reordering

Disadvantages of using symbolic registers

- Input code is not executable

Symbolic Registers in SLOTHY: Caveat

Caveat: Does not work with split heuristic (large pieces of code)

- Register allocation has to be done globally - if the combined scheduling/allocation problem is too hard you are out of luck
- Workaround: First perform register allocation (disallowing reordering), then perform scheduling using split heuristic

SLOTHY Basics: Hands-on Assignment 1 + 2

Installation

Option 1: Install from PyPI

```
python3 -m venv venv
source venv/bin/activate
pip install slothy
```

Installation

Option 1: Install from PyPI

```
python3 -m venv venv
source venv/bin/activate
pip install slothy
```

Option 2: Development Installation from Source

```
git clone https://github.com/slothy-optimizer/slothy.git
cd slothy
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Get Tutorial Files

Clone Tutorial Repository

```
$ git clone https://github.com/dop-amin/ches2025-slothy-tutorial
```

You will find five assignments

- `01basic/` (Basic – Using SLOTHY)
- `02basemul/` (ML-DSA basemul – Using symbolic registers and software pipelining)
- `03keccak/` (Keccak – Optimizing long code with SLOTHY)
- `04instruction/` (EON – Adding instructions to SLOTHY - **requires local SLOTHY clone**)
- `05fusion/` (EOR3 – Using SLOTHY's fusion feature)

Solutions will be presented as a part of this tutorial and can be requested by e-mail.

How to Use SLOTHY

Option 1: CLI

```
$ slothy-cli Arm_AArch64 \
    Arm_Cortex_A55 \
    input.s \
    -o output.s \
    -s start_label \
    -e end_label
```

Option 2: Python Script

```python
from slothy import Slothy
import slothy.targets.aarch64.aarch64_neon as AArch64_Neon
import slothy.targets.aarch64.cortex_a55 as Target_CortexA55

s = Slothy(AArch64_Neon, Target_CortexA55)
s.load_source_from_file("input.s")
s.optimize(start="start_label",
           end="end_label")
s.write_source_to_file("output.s")
```

- Vast majority of features available in both
- Today's assignments use Python scripts

Assignment 1: Polynomial Addition

Task
Optimize polynomial addition for ML-KEM using SLOTHY

```
void poly_add(int16_t *r, const int16_t *a, const int16_t *b)
```

- 256 16-bit coefficients
- AArch64 Neon assembly
- Target: Cortex-A55

Your Task: Complete the Python Script

`01basic/optimize.py`

```python
from slothy import Slothy
import slothy.targets.aarch64.aarch64_neon \
    as AArch64_Neon
import slothy.targets.aarch64.cortex_a55 \
    as Target_CortexA55

def main():
    # TODO: Initialize SLOTHY with AArch64_Neon
    # architecture and Target_CortexA55

    # TODO: Load the source assembly file

    # TODO: Optimize the code between
    # slothy_start and slothy_end markers

    # TODO: Write optimized code to output file

if __name__ == "__main__":
    main()
```

What You Need to Do

1. Create SLOTHY instance
2. Load `poly_add.s`
3. Call `optimize()` method
4. Save to `poly_add_opt_a55.s`

Key Files

- `poly_add.s` - input assembly
- `optimize.py` - your script

Input Assembly Structure

```
// From: 01basic/poly_add.s
mov x3, #4
loop:
  slothy_start:
  ldr q0, [x1], #128
  ldr q1, [x1, #-112]
  // ... (6 more loads from a)
  ldr q8, [x2], #128
  ldr q9, [x2, #-112]
  // ... (6 more loads from b)
  add v0.8h, v0.8h, v8.8h
  add v1.8h, v1.8h, v9.8h
  // ... (6 more adds)
  str q0, [x0], #128
  str q1, [x0, #-112]
  // ... (6 more stores)
  slothy_end:
  subs x3, x3, #1
  b.ne loop

```

Optimization Markers

- Start/end labels (any names work)
- Example: `slothy_start`, `slothy_end`

Manual Loop Unrolling

- Provides instruction-level parallelism
- Enables effective SLOTHY optimization

SLOTHY can also optimize loops

- Will be covered in detail later
- Advanced loop features:
  - Software pipelining
  - Automatic loop unrolling

Importing and Instantiating SLOTHY

Basic Import & Setup

```python
from slothy import Slothy

# Import architecture
import slothy.targets.aarch64.
    aarch64_neon
    as AArch64_Neon

# Import target microarchitecture
import slothy.targets.aarch64.
    cortex_a55
    as Target_CortexA55

# Create SLOTHY instance
s = Slothy(AArch64_Neon,
    Target_CortexA55)
```

Currently Available Targets

AArch64 (Neon):

- `cortex_a55`
- `cortex_a72_frontend`
- `apple_m1_firestorm_experimental`
- `apple_m1_icestorm_experimental`
- `neoverse_n1_experimental`
- `aarch64_big_experimental`

Arm v7-M:

- `cortex_m4`, `cortex_m7`

Arm v8.1-M (Helium):

- `cortex_m55r1`, `cortex_m85r1`

Core SLOTHY Methods

`load_source_from_file(filename)`

- Loads and parses assembly from file

Core SLOTHY Methods

`load_source_from_file(filename)`

- Loads and parses assembly from file

`write_source_to_file(filename)`

- Saves optimized assembly with metrics

Core SLOTHY Methods

`load_source_from_file(filename)`

- Loads and parses assembly from file

`write_source_to_file(filename)`

- Saves optimized assembly with metrics

`optimize(start=, end=)`

- **`start/end`** - optimization region markers
- Without markers, SLOTHY optimizes entire file
- Alternative: **`optimize_loop(looplabel)`** (see next assignment)

Full API documentation: **`slothy-optimizer.github.io/slothy/`**

Assignment 2: ML-DSA Basemul

Task
Use SLOTHY's software pipelining feature to optimize ML-DSA basemul

```
void poly_basemul_montgomery(int16_t *r, const int16_t *a, const int16_t
*b)
```

- Simple loop with symbolic registers
- AArch64 Neon assembly with Montgomery multiplication
- Target: Cortex-A55

Assignment 2: Input Assembly Structure

```
// From: 02basemul/basemul.s
modulus         .req v0
modulus_twisted .req v1
count           .req x3
t0              .req v7
.macro montgomery_reduce_long res, inl, inh
  uzp1   t0.4s, \inl\().4s, \inh\().4s
  mul    t0.4s, t0.4s, modulus_twisted.4s
  smlal  \inl\().2d, t0.2s, modulus.2s
  smlal2 \inh\().2d, t0.4s, modulus.4s
  uzp2   \res\().4s, \inl\().4s, \inh\().4s
.endm
.macro pmull dl, dh, a, b
  smull  \dl\().2d, \a\().2s, \b\().2s
  smull2 \dh\().2d, \a\().4s, \b\().4s
.endm
// ... (reg setup, modulus loading)
```

```
mov count, #256

loop_start:
  ldr q<aa>, [x1], #64
  ldr q<bb>, [x2], #64
  pmull v<resl>, v<resh>, v<aa>, v<bb>
  montgomery_reduce_long v<res>, v<resl>, v<resh>
  str q<res>, [x0], #64

  ldr q<aa>, [x1, #-48]
  ldr q<bb>, [x2, #-48]
  pmull v<resl>, v<resh>, v<aa>, v<bb>
  montgomery_reduce_long v<res>, v<resl>, v<resh>
  str q<res>, [x0, #-48]

  // ... (2 more similar iterations)
  subs count, count, #16
  cbnz count, loop_start
```

Optimizing Loops with SLOTHY

`optimize_loop(loop_lbl)`

- Optimizes a loop starting at a given label
- Automatically detects loop structure
- Example: `slothy.optimize_loop("loop_start")`

Optimizing Loops with SLOTHY

`optimize_loop(loop_lbl)`

- Optimizes a loop starting at a given label
- Automatically detects loop structure
- Example: `slothy.optimize_loop("loop_start")`

Supported Loop Patterns (AArch64)

- Counter decrement: `sub[s] <reg>, <reg>, #<imm>`
- Followed by conditional branch:
  - `cbnz/cbz <reg>, <loop_lbl>` (compare and branch)
  - `b.<cond> <loop_lbl>` (any Arm condition: ne, eq, ge, lt, etc.)

Optimizing Loops with SLOTHY

`optimize_loop(loop_lbl)`

- Optimizes a loop starting at a given label
- Automatically detects loop structure
- Example: `slothy.optimize_loop("loop_start")`

Supported Loop Patterns (AArch64)

- Counter decrement: `sub[s] <reg>, <reg>, #<imm>`
- Followed by conditional branch:
  - `cbnz/cbz <reg>, <loop_lbl>` (compare and branch)
  - `b.<cond> <loop_lbl>` (any Arm condition: ne, eq, ge, lt, etc.)
- **Note:** Loop patterns are extensible - additional patterns can be added to the architecture model

Optimizing Loops with SLOTHY

`optimize_loop(loop_lbl)`

- Optimizes a loop starting at a given label
- Automatically detects loop structure
- Example: `slothy.optimize_loop("loop_start")`

Supported Loop Patterns (AArch64)

- Counter decrement: `sub[s] <reg>, <reg>, #<imm>`
- Followed by conditional branch:
  - `cbnz/cbz <reg>, <loop_lbl>` (compare and branch)
  - `b.<cond> <loop_lbl>` (any Arm condition: ne, eq, ge, lt, etc.)
- **Note:** Loop patterns are extensible - additional patterns can be added to the architecture model

Limitations

- Loop counter must be decremented by a constant
- Loop body must be straight-line code (no branches inside)
- Subtraction must immediately precede the branch instruction

SLOTHY Configuration Options

Setting Configuration Options

Configuration options are set after creating the SLOTHY instance:

Example: Setting Configuration Options

```
# Create SLOTHY instance
slothy = Slothy(Architecture, Target)

# Configure SLOTHY settings
slothy.config.<option1> = <value1>
slothy.config.<option2> = <value2>
slothy.config.<option3>.<suboption> = <value3>

# ... load, optimize, write

```

Full configuration reference:

`slothy-optimizer.github.io/slothy/apidocs/slothy/slothy.core.config.html`

Software Pipelining Configuration i

`slothy.config.sw_pipelining.enabled`

Enable software pipelining optimization for loop code blocks.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Essential for optimizing loops with overlapping iterations

Software Pipelining Configuration ii

`slothy.config.sw_pipelining.allow_pre`

Allow 'early' instructions to be pulled forward from future iterations.

- **Default:** `True`
- **Type:** Boolean (`True|False`)
- **Usage:** Enables forward movement of instructions from iteration N+1 to N

Software Pipelining Configuration iii

`slothy.config.sw_pipelining.allow_post`

Allow 'late' instructions to be deferred to later iterations.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Enables backward movement of instructions from iteration N to N+1

Software Pipelining Configuration iv

`slothy.config.sw_pipelining.unroll`

The unrolling factor to use for software pipelining optimization.

- **Default:** `1`
- **Type:** Integer (positive number)
- **Usage:** Higher values enable more aggressive pipelining optimizations

Software Pipelining Configuration v

`slothy.config.sw_pipelining.optimize_preamble`

Perform a separate optimization pass for the loop preamble section.

- **Default:** `True`
- **Type:** Boolean (`True|False`)
- **Usage:** Optimizes setup instructions before the main pipelined loop

Software Pipelining Configuration vi

`slothy.config.sw_pipelining.optimize_postamble`

Perform a separate optimization pass for the loop postamble section.

- **Default:** `True`
- **Type:** Boolean (`True|False`)
- **Usage:** Optimizes cleanup instructions after the main pipelined loop

Performance Configuration

`slothy.config.variable_size`

Model the number of stalls as a parameter in the constraint satisfaction problem.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Enable for small/medium code where solver can minimize stalls directly; if `False`: External binary search finds minimum stalls

`slothy.config.constraints.stalls_first_attempt`

The initial number of stalls to attempt during optimization.

- **Default:** `0`
- **Type:** Integer (non-negative number)
- **Usage:** Set higher when minimum stall count is known from experience

Input/Output Configuration

`slothy.config.outputs`

List of architectural registers that must be preserved as function outputs.

- **Default:** `[]`
- **Type:** List of strings (register names or `flags`)
- **Usage:** Critical for correctness - prevents clobbering output registers

`slothy.config.inputs_are_outputs`

Treat all input registers as outputs that must be preserved.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Typically used for loop optimization

Register Management Configuration

`slothy.config.reserved_regs`

Set of architectural registers excluded from register renaming and allocation.

- **Default:** `["sp"]` (AArch64)
- **Type:** List of strings (register names)
- **Note:** Overwrites default reserved registers for the target architecture

Note: Reserved registers are ”locked” by default - they won’t be introduced during renaming, but existing uses won’t be touched either.

Hands-on Exercise

[30 minutes hands-on exercise (Assignment 1 and 2)]

See the `README.md` for hints

Tutorial Assignments

github.com/dop-amin/ches2025-slothy-tutorial

Tutorial Slides

kannwischer.eu/talks/20250914_slothy.pdf

Assignment 1: Solution

01basic/optimize.py

```python
from slothy import Slothy
import slothy.targets.aarch64.aarch64_neon as AArch64_Neon
import slothy.targets.aarch64.cortex_a55 as Target_CortexA55

def main():
    # Initialize SLOTHY
    slothy = Slothy(AArch64_Neon, Target_CortexA55)

    # Load the source assembly file
    slothy.load_source_from_file("poly_add.s")

    # Optimize the code between markers
    slothy.optimize(start="slothy_start", end="slothy_end")

    # Write optimized code to output file
    slothy.write_source_to_file("poly_add_opt_a55.s")

if __name__ == "__main__":
    main()
```

Assignment 1: Optimized Result

```
slothy_start:
                                    // Instructions:    32
                                    // Expected cycles: 49
                                    // Expected IPC:    0.65
                                    //
                                    // Cycle bound:     49.0
                                    // IPC bound:       0.65
                                    //
                                    // Wall time:     0.07s
                                    // User time:     0.07s
                                    //
                                    // --------------- cycle (expected) --------------->
                                    // 0                        25
                                    // |------------------------|-----------------------
    ldr q27, [x1], #128             // *................................................
    ldr q8, [x2], #128              // ..*..............................................
    ldr q30, [x1, #-112]            // ....*............................................
    add v9.8H, v27.8H, v8.8H        // ......*..........................................
    ldr q23, [x2, #-112]            // .......*.........................................
    ldr q1, [x2, #-96]              // .........*.......................................
    add v4.8H, v30.8H, v23.8H       // ...........*.....................................
    str q9, [x0], #128              // ............*....................................
    ldr q15, [x1, #-96]             // .............*...................................
    ldr q10, [x1, #-80]             // ...............*.................................
    ldr q7, [x2, #-80]              // .................*...............................
    add v0.8H, v15.8H, v1.8H        // ...................*.............................
    // ... (continues)
slothy_end:
```

Assignment 2: Solution

02basemul/optimize.py

```python
from slothy import Slothy
import slothy.targets.aarch64.aarch64_neon as AArch64_Neon
import slothy.targets.aarch64.cortex_a55 as Target_CortexA55

def main():
    # Initialize SLOTHY
    slothy = Slothy(AArch64_Neon, Target_CortexA55)

    # Enable software pipelining
    slothy.config.sw_pipelining.enabled = True
    slothy.config.variable_size = True
    slothy.config.constraints.stalls_first_attempt = 32

    # Load source and optimize loop
    slothy.load_source_from_file("basemul.s")
    slothy.optimize_loop("loop_start")
    slothy.write_source_to_file("basemul_opt_a55.s")

if __name__ == "__main__":
    main()
```

Assignment 2: Optimized Result

```
        ldr q7, [x2, #32]        // *.............................
        sub count, count, #16
loop_start:
                                             // Instructions:    40
                                             // Expected cycles: 48
                                             // Expected IPC:    0.83
                                             //
                                             // Cycle bound:     21.0
                                             // IPC bound:       1.90
                                             //
                                             // Wall time:     5.36s
                                             // User time:     5.36s
                                             //
                                             // -------------- cycle (expected) --------------->
                                             // 0                        25
                                             // |------------------------|----------------------
        ldr q23, [x2], #64                   // *...............................................
        ldr q10, [x1], #64                   // ..*.............................................
        ldr q4, [x1, #-32]                   // ....*...........................................
        smull v2.2D, v10.2S, v23.2S          // ......*.........................................
        smull2 v13.2D, v10.4S, v23.4S        // .......*........................................
        smull v5.2D, v4.2S, v7.2S            // ........*.......................................
        smull2 v14.2D, v4.4S, v7.4S          // .........*......................................
        ldr q4, [x1, #-16]                   // ..........*.....................................
        uzp1 v31.4S, v2.4S, v13.4S           // ............*...................................
        ldr q24, [x2, #-16]                  // .............*..................................
        // ... (continues)
```

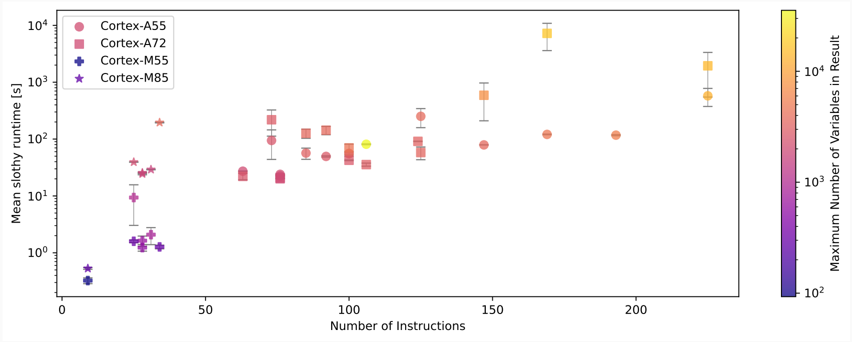
Did our code actually get faster?

Assignment	Before (cycles)	After (cycles)	Speedup
1) Poly Add (A55)	228	204	$1.12\times$
2) Poly Basemul (A55)	1512	822	$1.84\times$

Table 1: Performance on Cortex-A55 cores

Heuristics and Register Spilling

Why Heuristics?



Hardness of the problems SLOTHY deals with depends on a number of factors:

- Number of instructions in the assembly
- Complexity of μArch model
- Is software pipelining/spilling wanted?

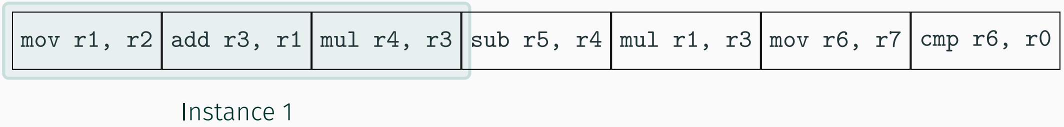
Splitting Heuristic

Usually: SLOTHY considers the input code as one, large chunk. Optimizing this chunk may not terminate if there are too many instructions (i.e., the problem gets too hard).

Splitting Heuristic

Usually: SLOTHY considers the input code as one, large chunk. Optimizing this chunk may not terminate if there are too many instructions (i.e., the problem gets too hard).

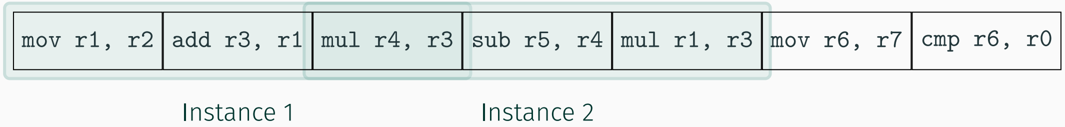
Idea: Pick a window containing a modest number of instructions and slide this window across the code, then repeat.



Splitting Heuristic

Usually: SLOTHY considers the input code as one, large chunk. Optimizing this chunk may not terminate if there are too many instructions (i.e., the problem gets too hard).

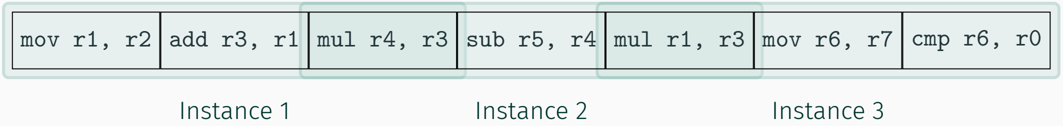
Idea: Pick a window containing a modest number of instructions and slide this window across the code, then repeat.



Splitting Heuristic

Usually: SLOTHY considers the input code as one, large chunk. Optimizing this chunk may not terminate if there are too many instructions (i.e., the problem gets too hard).

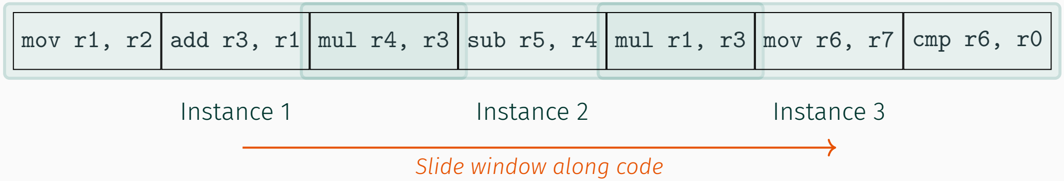
Idea: Pick a window containing a modest number of instructions and slide this window across the code, then repeat.



Splitting Heuristic

Usually: SLOTHY considers the input code as one, large chunk. Optimizing this chunk may not terminate if there are too many instructions (i.e., the problem gets too hard).

Idea: Pick a window containing a modest number of instructions and slide this window across the code, then repeat.



Splitting Heuristic: Useful Options i

`slothy.config.split_heuristic`

Trade-off between runtime and optimality by splitting code blocks into subchunks.

- **Default:** `False`
- **Type:** Boolean (`True`|`False`)
- **Usage:** Enable when optimization of large code blocks fails

Splitting Heuristic: Useful Options ii

`slothy.config.split_heuristic_factor`

Number of subchunks to split each code block into for optimization.

- **Default:** `2`
- **Type:** Integer (positive number)
- **Usage:** Only meaningful when `split_heuristic=True`

Splitting Heuristic: Useful Options iii

`slothy.config.split_heuristic_stepsize`

Increment for the sliding window as fraction of total code size.

- **Default:** `None`
- **Type:** Float (0.0 to 1.0)
- **Usage:** Controls overlap between optimization windows

Splitting Heuristic: Useful Options iv

slothy.config.split_heuristic_repeat

Number of times the splitting heuristic procedure should be repeated.

- **Default:** `1`
- **Type:** Integer (positive number)
- **Usage:** Facilitate more interleaving; improves output code

Interleaving Heuristic i

Whenever the code-paths to be interleaved are very far apart, establishing a coarse interleaving on the CFG can be helpful to facilitate later optimization.

`slothy.config.split_heuristic_preprocess_naive_interleaving`

Interleave instructions by lowest depth without applying register renaming.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Helps when code paths to be interleaved are far apart

Interleaving Heuristic ii

`slothy.config.split_heuristic_preprocess_naive_interleaving_strategy`

Strategy for naive interleaving preprocessing of distant code paths.

- **Default:** `"depth"`
- **Type:** String (`"depth"|"alternate"`)
- **Usage:** Choose interleaving pattern for coarse CFG optimization

Spill-code Generation

Problem: When writing a symbolic implementation, register allocation is unclear.

Spill-code Generation

Problem: When writing a symbolic implementation, register allocation is unclear.

- How many registers will be required?
- Are there enough registers?
- What are we going to do when we run out of registers?

Spill-code generation

Solution: Let SLOTHY take care of this. SLOTHY can …

- …introduce spills/restores whenever it runs out of registers.

Spill-code generation

Solution: Let SLOTHY take care of this. SLOTHY can …

- …introduce spills/restores whenever it runs out of registers.
- …minimize the number of spills it introduces as an optimization target.

Spill-code generation

Solution: Let SLOTHY take care of this. SLOTHY can …

- …introduce spills/restores whenever it runs out of registers.
- …minimize the number of spills it introduces as an optimization target.
- …absorb superfluous spills in existing, non-symbolic code in case a register allocation without these spills exists.

Spill-code generation

Solution: Let SLOTHY take care of this. SLOTHY can ...

- ...introduce spills/restores whenever it runs out of registers.
- ...minimize the number of spills it introduces as an optimization target.
- ...absorb superfluous spills in existing, non-symbolic code in case a register allocation without these spills exists.
- ...*not* minimize spills AND reorder instructions simultaneously $\rightarrow$ State explosion

Useful Options for Spilling i

`slothy.config.allow_spills`

Allow SLOTHY to introduce stack spills when register pressure is too high.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Essential for symbolic assembly with high register pressure

Useful Options for Spilling ii

`slothy.config.minimize_spills`

Minimize the number of stack spills as the solver's optimization objective.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Enable when spill count minimization is the primary goal

Useful Options for Spilling iii

`slothy.config.absorb_spills`

Remove existing spills from non-symbolic code when better register allocation exists. Does *not* work with splitting heuristic.

- **Default:** `True`
- **Type:** Boolean (`True|False`)
- **Usage:** Cleanup existing assembly with unnecessary spills

Useful Options for Spilling iv

`slothy.config.constraints.functional_only`

Disable modeling of latencies and functional units.

- **Default:** `False`
- **Type:** Boolean (`True|False`)
- **Usage:** Reduce complexity of the optimization problem when, e.g., focusing on register allocation

Useful Options for Spilling v

`slothy.config.constraints.allow_reordering`

Allow reordering of instructions.

- **Default:** `True`
- **Type:** Boolean (`True|False`)
- **Usage:** Reduce complexity of the optimization problem when, e.g., focusing on register allocation

Memory Dependencies

```
ldr X<count>, [sp, #STACK_OFFSET_COUNT] //
     @slothy:reads=STACK_OFFSET_COUNT
ldr X<const>, [X<caddr>, X<count>, LSL #3]
add X<count>, X<count>, #1
cmp X<count>, #(KECCAK_F1600_ROUNDS-1)
str X<count>, [sp, #STACK_OFFSET_COUNT] //
     @slothy:writes=STACK_OFFSET_COUNT

```

- Reads/writes marked through `@slothy` tags
- Enables safely re-scheduling interdependent reads/writes
- Internally: “hint”-registers

Assignment 3: Keccak Permutation

Task
Use SLOTHY's spill-code generation & splitting heuristic to optimize the Keccak permutation.

```
void keccak_f1600_x4_v8a_hybrid_slothy_symbolic(void* states)
```

- Symbolic implementation; not enough registers
- Large number of instructions
- AArch64 Scalar+Neon hybrid assembly
- Target: Cortex-A55

Assignment 3: Input Assembly

```
// Many macros...
keccak_f1600_x4_v8a_hybrid_slothy_symbolic:
    // Some code
 initial:
    scalar_round_initial    // @slothy:interleaving_class=0
    scalar_round_noninitial // @slothy:interleaving_class=0
    vector_round            // @slothy:interleaving_class=1
 loop:
    scalar_round_noninitial // @slothy:interleaving_class=0
    scalar_round_noninitial // @slothy:interleaving_class=0
    vector_round            // @slothy:interleaving_class=1
 loop_end:
    ble loop
    // More code
    b initial
done:
    // More code
    ret

```

- Two intervals to optimize
- `// @slothy:...` tags
  - → Eased coarse interleaving
  - → Memory dependencies

Assignment 3: Python Template

```python
from slothy import Slothy
import slothy.targets.aarch64.aarch64_neon as AArch64_Neon
import slothy.targets.aarch64.cortex_a55 as Target_CortexA55

def main():
    slothy = Slothy(AArch64_Neon, Target_CortexA55)
    # Load the source assembly file
    slothy.load_source_from_file("keccak.s")
    # ...
    # --- Register Allocation
    # TODO: complete
    slothy.write_source_to_file("keccak_alloc_a55.s")
    # ...
    # --- Optimize
    slothy.load_source_from_file("keccak_alloc_a55.s")
    # TODO: complete
    # Write optimized code to output file
    slothy.write_source_to_file("keccak_opt_a55.s")
```

Two separate goals for optimization

- Register allocation
- Instruction Scheduling

Hands-on Exercise

[30 minutes hands-on exercise (Assignment 3)]

See the `README.md` for hints

Tutorial Assignments
github.com/dop-amin/ches2025-slothy-tutorial

Tutorial Slides
kannwischer.eu/talks/20250914_slothy.pdf

Assignment 3: Solution, Setup

03keccak/optimize.py

```python
from slothy import Slothy
import slothy.targets.aarch64.aarch64_neon as AArch64_Neon
import slothy.targets.aarch64.cortex_a55 as Target_CortexA55

def main():
    # Initialize SLOTHY
    slothy = Slothy(AArch64_Neon, Target_CortexA55)
    # Load the source assembly file
    slothy.load_source_from_file("keccak.s")
    # Common config
    slothy.config.selftest = False
    slothy.config.timeout = 180
    slothy.config.constraints.stalls_first_attempt = 32
    slothy.config.reserved_regs = ["sp"]
    slothy.config.outputs = ["hint_STACK_OFFSET_COUNT"]  # preserve count
    slothy.config.inputs_are_outputs = True
    slothy.config.variable_size = True
    slothy.config.with_preprocessor = True
    common_conf = slothy.config.copy()
```

Assignment 3: Solution, Register Allocation

03keccak/optimize.py

```
    slothy.config.constraints.functional_only = True
    slothy.config.constraints.allow_reordering = False
    slothy.config.constraints.allow_spills = True
    slothy.config.constraints.minimize_spills = True

    # Call to the optimizer
    slothy.optimize(start="loop", end="loop_end")
    slothy.optimize(start="initial", end="loop")

    slothy.write_source_to_file("keccak_alloc_a55.s")

    # Reset configuration to common
    slothy.config = common_conf.copy()
```

Assignment 3: Solution, Scheduling

03keccak/optimize.py

```python
slothy.load_source_from_file("keccak_alloc_a55.s")

# Configure optimization parameters
slothy.config.split_heuristic = True
slothy.config.split_heuristic_preprocess_naive_interleaving = True
slothy.config.split_heuristic_preprocess_naive_interleaving_strategy = "alternate"

slothy.config.split_heuristic_factor = 12
slothy.config.split_heuristic_repeat = 2
slothy.config.split_heuristic_stepsize = 0.05
slothy.config.absorb_spills = False  # Does not work with splitting

# Call to the optimizer
slothy.optimize(start="loop", end="loop_end")
slothy.optimize(start="initial", end="loop")

# Write optimized code to output file
slothy.write_source_to_file("keccak_opt_a55.s")
```

Assignment 3: Performance

	Before		After		
Assignment	Cycles	IPC	Cycles	IPC	Speedup
Keccak	8212	1.18	5385	1.81	1.53×

Table 2: Keccak performance on Cortex-A55 cores

Modelling Architectures & Microarchitectures

Extending SLOTHY: Architecture & Microarchitecture Models

Motivation

- SLOTHY is built to be extensible
- We would love to see new architectures or extensions to existing ones

Extending SLOTHY: Architecture & Microarchitecture Models

Motivation

- SLOTHY is built to be extensible
- We would love to see new architectures or extensions to existing ones

When You'll Need This Knowledge

- Adding support for a **new architecture** (e.g., RISC-V)
- Adding a **new microarchitecture** (e.g., Arm Cortex-A510)
- Adding **missing instructions** to existing architecture or microarchitecture models
- **Fixing or refining** performance characteristics

Extending SLOTHY: Architecture & Microarchitecture Models

Motivation

- SLOTHY is built to be extensible
- We would love to see new architectures or extensions to existing ones

When You'll Need This Knowledge

- Adding support for a **new architecture** (e.g., RISC-V)
- Adding a **new microarchitecture** (e.g., Arm Cortex-A510)
- Adding **missing instructions** to existing architecture or microarchitecture models
- **Fixing or refining** performance characteristics

The Reality of SLOTHY Models

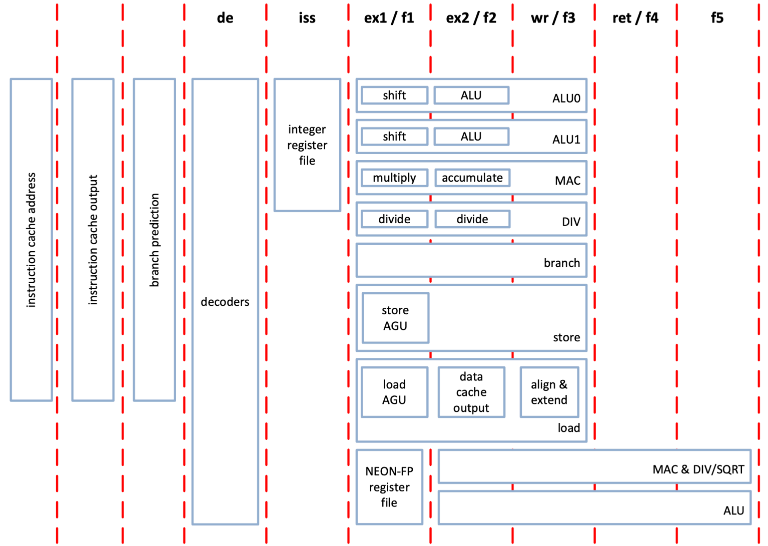
- SLOTHY's architecture and microarchitecture models are **lazily built**
- Only commonly used instructions are modeled
- Models grow as users need them (AArch64 is the most mature)

Understanding the Microarchitecture

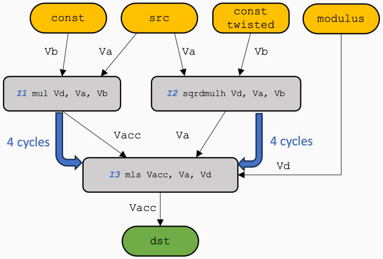
Starting Point: Software Optimization Guides (SWOG)

- Manufacturers often provide optimization guides
- Contains pipeline diagrams, latency tables, and throughput information
- Example: Cortex-A55 `https://developer.arm.com/documentation/EPM128372/latest/`

Cortex-A55 Pipeline



Scheduling as constraint solving problem: Latencies



Instruction group	AArch64 instructions	Exec latency	Execution throughput
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4	2*

* If the instruction has Q-form, the Q-form of the instruction can only be dual issued as instruction 0 and execution throughput is 1.

- Q-form: `mul v0.4s, v1.4s, v2.4s` (128-bit operation)
- D-form: `mul v0.2s, v1.2s, v2.2s` (64-bit operation)

Constraint Model Intuition

How constraint models are built from the microarchitecture model

Note: You don’t need to implement this yourself - SLOTHY handles it internally.

Each instruction `I` is assigned:

- Functional unit(s): `Unit(I)` - where it executes
- Usage time: `block(I)` - how long each unit is occupied (inverse throughput)
- Latency: `lat(I)` - when results are available

Constraint Model Intuition

How constraint models are built from the microarchitecture model

Note: You don't need to implement this yourself - SLOTHY handles it internally.

Each instruction `I` is assigned:

- **Functional unit(s):** `Unit(I)` - where it executes
- **Usage time:** `block(I)` - how long each unit is occupied (inverse throughput)
- **Latency:** `lat(I)` - when results are available

Latency $\neq$ Usage Time

- Latency can be **smaller** or **larger** than usage time
- Example: `VADD` on Cortex-M55:
  - Occupies vector unit for 2 cycles (usage time)
  - Result available after 1 cycle (latency)

Core Microarchitectural Constraints

1. Latency Constraints
    - Consumer must wait for producer's results
    - `consumer.pos` $\geq$ `producer.pos + latency`

Core Microarchitectural Constraints

1. Latency Constraints
    - Consumer must wait for producer's results
    - `consumer.pos` $\geq$ `producer.pos + latency`
2. Execution Unit + Throughput Constraints
    - Instructions cannot overlap on the same unit
    - For each unit maintain a list of usage intervals: `[pos, pos + inverse_throughput]`

Core Microarchitectural Constraints

1. Latency Constraints
    - Consumer must wait for producer's results
    - `consumer.pos` $\geq$ `producer.pos + latency`
2. Execution Unit + Throughput Constraints
    - Instructions cannot overlap on the same unit
    - For each unit maintain a list of usage intervals: `[pos, pos + inverse_throughput]`

Warning: SWOG and SLOTHY express throughput differently!
SWOG shows overall throughput, we need inverse throughput per unit!
Commonly:

- SWOG: "Throughput = 8" $\Rightarrow$ usually 8 units $\Rightarrow$ SLOTHY: inverse throughput = 1
- SWOG: "Throughput = 1/2" $\Rightarrow$ 1 unit, busy for 2 cycles $\Rightarrow$ SLOTHY: inverse throughput = 2

$\Longrightarrow \quad \text{tp}_{\text{SLOTHY}} = \frac{\#\text{units}}{\text{tp}_{\text{SWOG}}}$

Modeling Q-form vs D-form Throughput

Special case of the Arm Cortex-A55

- D-form (64-bit): throughput = 2 (can dual-issue)
- Q-form (128-bit): throughput = 1 (single-issue only)

Modeling Q-form vs D-form Throughput

Special case of the Arm Cortex-A55

- D-form (64-bit): throughput = 2 (can dual-issue)
- Q-form (128-bit): throughput = 1 (single-issue only)

SLOTHY's Solution: Model the vector unit as two virtual units: VEC0 and VEC1

- **D-form instructions:** Occupy either VEC0 **or** VEC1
- **Q-form instructions:** Occupy **both** VEC0 **and** VEC1

Modeling Q-form vs D-form Throughput

Special case of the Arm Cortex-A55

- D-form (64-bit): throughput = 2 (can dual-issue)
- Q-form (128-bit): throughput = 1 (single-issue only)

SLOTHY's Solution: Model the vector unit as two virtual units: VEC0 and VEC1

- **D-form instructions:** Occupy either VEC0 **or** VEC1
- **Q-form instructions:** Occupy **both** VEC0 **and** VEC1

Keep in mind: SLOTHY models are approximate!

- We do not try to model each aspect of the microarchitecture.
- **Goal**: Simple enough model that achieves good performance!
- Above's example: Currently SLOTHY does not model the D-form of multiplications as we did not need it so far.

Example Walkthrough: Neon Multiplication

What We'll Cover
Walk through how the Neon `mul` instruction is modeled in SLOTHY:

1. Architecture model (instruction syntax and semantics)
2. Microarchitecture model (performance characteristics)
3. How they work together

Architecture Model: `vmul` Instruction

`slothy/targets/aarch64/aarch64_neon.py`

```python
class vmul(AArch64Instruction):
    pattern = "mul <Vd>.<dt0>, <Va>.<dt1>, <Vb>.<dt2>"
    inputs = ["Va", "Vb"]
    outputs = ["Vd"]

```

What This Defines

- **Pattern:** How to parse/emit the instruction
- **Inputs:** Source operands (Va, Vb)
- **Outputs:** Destination operand (Vd)
- **Placeholders:** `<dt0>, <dt1>, <dt2>` for data types (e.g., `4s`, `8h`)
- Parsing complexity is hidden away in `AArch64Instruction` class

Microarchitecture Model Structure

slothy/targets/aarch64/cortex_a55.py - Basic Structure

```python
class ExecutionUnit(Enum):
    SCALAR_ALU0 = 1
    SCALAR_ALU1 = 2
    SCALAR_MAC = 3
    SCALAR_LOAD = 4
    SCALAR_STORE = 5
    VEC0 = 6
    VEC1 = 7

execution_units = {
    # Map instruction classes to execution units
    # ...
}

inverse_throughput = {
    # Map instruction classes to inverse throughput
    # ...
}

default_latencies = {
    # Map instruction classes to result latency
    # ...
}

```

Microarchitecture Model: `vmul` Instruction

Adding Performance Characteristics

```
# From slothy/targets/aarch64/cortex_a55.py
execution_units = {
    # ...
    vmul: [[ExecutionUnit.VEC0, ExecutionUnit.VEC1]], # Q-form uses both VEC0 and VEC1
}

inverse_throughput = {
    # ...
    vmul: 1, # Occupies units for 1 cycle
}

default_latencies = {
    # ...
    vmul: 4, # Result available after 4 cycles
}
```

Alternative: Modeling Q-form vs D-form

```
# More precise (not currently in model):
is_qform_form_of(vmul): [[ExecutionUnit.VEC0, ExecutionUnit.VEC1]],  # VEC0 AND VEC1
is_dform_form_of(vmul): [ExecutionUnit.VEC0, ExecutionUnit.VEC1],  # VEC0 OR VEC1
```

Grouping Instructions with Class Hierarchies

Architecture Model: Using Inheritance for Groups

```python
# slothy/targets/aarch64/aarch64_neon.py
class AArch64NeonLogical(AArch64Instruction):
    pass

class veor(AArch64NeonLogical):
    pattern = "eor <Vd>.<dt0>, <Va>.<dt1>, <Vb>.<dt2>"
    inputs = ["Va", "Vb"]
    outputs = ["Vd"]

class vbic(AArch64NeonLogical):
    pattern = "bic <Vd>.<dt0>, <Va>.<dt1>, <Vb>.<dt2>"
    inputs = ["Va", "Vb"]
    outputs = ["Vd"]
```

Grouping Instructions with Class Hierarchies

Architecture Model: Using Inheritance for Groups

```python
# slothy/targets/aarch64/aarch64_neon.py
class AArch64NeonLogical(AArch64Instruction):
    pass

class veor(AArch64NeonLogical):
    pattern = "eor <Vd>.<dt0>, <Va>.<dt1>, <Vb>.<dt2>"
    inputs = ["Va", "Vb"]
    outputs = ["Vd"]

class vbic(AArch64NeonLogical):
    pattern = "bic <Vd>.<dt0>, <Va>.<dt1>, <Vb>.<dt2>"
    inputs = ["Va", "Vb"]
    outputs = ["Vd"]
```

Microarchitectural Model: Assign Properties to Groups

```python
# slothy/targets/aarch64/cortex_a55.py
default_latencies = {
    # ...
    AArch64NeonLogical: 1
}
```

Microarchitectural Model: Forwarding Paths

Fast Result Forwarding

Some CPUs have special paths that allow results to be used faster in specific cases

Example: Cortex-A72 mla→mla Forwarding

```python
# slothy/targets/aarch64/cortex_a72_frontend.py
def get_latency(src, out_idx, dst):
    # Default latency (e.g., vmla has latency 5)
    latency = lookup_multidict(default_latencies, src)

    # forwarding paths and other special classes
    instclass_src = find_class(src)
    instclass_dst = find_class(dst)

    # Fast mla->mla forwarding: reduces latency to 1
    if (instclass_src == vmla and instclass_dst == vmla
        and src.args_in_out[0] == dst.args_in_out[0]):  # Same accumulator
        return 1  # Instead of 5!

    return latency

```

Microarchitectural Model: Slot Constraints

Issue Restrictions
Some instructions can only issue in specific issue slots

Example: Cortex-A55 Q-form Restrictions

```python
# slothy/targets/aarch64/cortex_a55.py
def add_further_constraints(slothy):
    # fcsel
    slothy.restrict_slots_for_instructions_by_class(
        [fcsel_dform], [0]
    )

    # Q-form vector instructions on slot 0 only
    slothy.restrict_slots_for_instructions_by_property(
        Instruction.is_q_form_vector_instruction, [0]
    )

```

Instruction Fusion & Splitting

Instruction Fusion & Splitting

Two complementary optimization techniques:

Instruction Fusion & Splitting

Two complementary optimization techniques:

- **Instruction Fusion**: Combine multiple “simple” instructions into one complex instruction

Instruction Fusion & Splitting

Two complementary optimization techniques:

- **Instruction Fusion**: Combine multiple “simple” instructions into one complex instruction
- **Instruction Splitting**: Break “complex” instructions into simpler ones for better scheduling

Instruction Fusion & Splitting

Two complementary optimization techniques:

- **Instruction Fusion**: Combine multiple “simple” instructions into one complex instruction
- **Instruction Splitting**: Break “complex” instructions into simpler ones for better scheduling

Idea: Allow certain simple instruction replacements

Different microarchitectures have different capabilities. Let SLOTHY adapt code to better utilize target-specific features!

Example: Instruction Splitting on Cortex-M7 (dual issue)

Without Splitting (12 cycles):

```
ldm r0, {r2-r9}     // *...........
uadd16 r8, r8, r1   // ....*.......
uadd16 r5, r5, r1   // .....*......
uadd16 r7, r7, r1   // ......*.....
uadd16 r6, r6, r1   // .......*....
uadd16 r4, r4, r1   // ........*...
uadd16 r3, r3, r1   // .........*..
uadd16 r2, r2, r1   // ..........*.
uadd16 r9, r9, r1   // ...........*
```

Example: Instruction Splitting on Cortex-M7 (dual issue)

Without Splitting (12 cycles):

```
ldm r0, {r2-r9}     // *...........
uadd16 r8, r8, r1   // ....*.......
uadd16 r5, r5, r1   // .....*......
uadd16 r7, r7, r1   // ......*.....
uadd16 r6, r6, r1   // .......*....
uadd16 r4, r4, r1   // ........*...
uadd16 r3, r3, r1   // .........*..
uadd16 r2, r2, r1   // ..........*.
uadd16 r9, r9, r1   // ...........*
```

With Splitting (9 cycles):

```
ldr r10, [r0, #0]    // *........
uadd16 r2, r10, r1   // .*.......
ldr r11, [r0, #28]   // .*.......
uadd16 r9, r11, r1   // ..*......
ldr r14, [r0, #4]    // ..*......
uadd16 r3, r14, r1   // ...*.....
ldr r4, [r0, #8]     // ...*.....
// ... (dual-issued)
```

Performance Impact

33% speedup by enabling dual-issue of loads and arithmetic operations!

Splitting Callback

```
def ldm_interval_splitting_cb():
  def core(inst, t, log=None):
    ptr = inst.args_in[0]
    regs = inst.args_out
    width = inst.width
    t.inst = []
    offset = 0
    for r in regs:
      ldr = Armv7mInstruction.build(
        ldr_with_imm, {"width": width, "Rd": r, "Ra": ptr, "imm": f"#{off}"})
      ldr.pre_index = offset
      t.inst.append(ldr)
      offset += 4
      ldr_src = (SourceLine(ldr.write()).add_tags(inst.source_line.tags).add_comments(inst.source_line.comments))
      ldr.source_line = ldr_src
    t.changed = True
    return True
  return core
```

Splitting & Fusion: Limitations

- Splitting/Fusion can either be enabled/disabled; SLOTHY is not able to determine dynamically if applying the heuristic would be beneficial
  - $\rightarrow$ State explosion

Splitting & Fusion: Limitations

- Splitting/Fusion can either be enabled/disabled; SLOTHY is not able to determine dynamically if applying the heuristic would be beneficial
  - $\rightarrow$ State explosion
- Splitting/Fusion is not guaranteed to succeed, e.g., in case of the transformation requiring more registers

Splitting & Fusion: Limitations

- Splitting/Fusion can either be enabled/disabled; SLOTHY is not able to determine dynamically if applying the heuristic would be beneficial
  - $\rightarrow$ State explosion
- Splitting/Fusion is not guaranteed to succeed, e.g., in case of the transformation requiring more registers
- The developer of a callback is responsible for the safety and security of the transformation
  - $\rightarrow$ Not covered by SLOTHY's selfcheck

Splitting & Fusion: Limitations

- Splitting/Fusion can either be enabled/disabled; SLOTHY is not able to determine dynamically if applying the heuristic would be beneficial
  - $\rightarrow$ State explosion
- Splitting/Fusion is not guaranteed to succeed, e.g., in case of the transformation requiring more registers
- The developer of a callback is responsible for the safety and security of the transformation
  - $\rightarrow$ Not covered by SLOTHY's selfcheck
- **Current Limitation**: Splitting/Fusion can only enabled/disabled globally.

Assignment 4: Adding Instructions to Architecture & Microarchitecture Models

Your Task

Add support for the AArch64 `eon` (Exclusive OR NOT) instruction to SLOTHY

- Operation: `Xd = Xa XOR NOT(Xb)`
- Example: `eon x2, x2, x10`

Assignment 4: Adding Instructions to Architecture & Microarchitecture Models

Your Task

Add support for the AArch64 `eon` (Exclusive OR NOT) instruction to SLOTHY

- Operation: `Xd = Xa XOR NOT(Xb)`
- Example: `eon x2, x2, x10`

Steps

1. **Architecture Model:** Teach SLOTHY to parse `eon` instructions
   `slothy/targets/aarch64/aarch64_neon.py`
2. **Microarchitecture Model:** Add performance characteristics for Cortex-A55
   `slothy/targets/aarch64/cortex_a55.py`

Assignment 4: Adding Instructions to Architecture & Microarchitecture Models

Your Task

Add support for the AArch64 `eon` (Exclusive OR NOT) instruction to SLOTHY

- Operation: `Xd = Xa XOR NOT(Xb)`
- Example: `eon x2, x2, x10`

Steps

1. **Architecture Model:** Teach SLOTHY to parse `eon` instructions
   `slothy/targets/aarch64/aarch64_neon.py`
2. **Microarchitecture Model:** Add performance characteristics for Cortex-A55
   `slothy/targets/aarch64/cortex_a55.py`

Getting Started

- See files in `04instruction/`
- Read the `README.md` for detailed instructions and hints

Assignment 4: Setup Instructions

Clone SLOTHY

- This assignment requires modifying SLOTHY source code
  $\Longrightarrow$ We need to work on a local clone of SLOTHY
- The assignment code checks that you are not accidentally using SLOTHY from pip

Setup

1. `cd 04instruction/`
2. `git clone https://github.com/slothy-optimizer/slothy.git`
3. `python3 -m venv venv`
4. `source venv/bin/activate`
5. `pip3 install -r slothy/requirements.txt`
6. Test: `python3 optimize.py` (should fail with parsing error)

Assignment 4: `test_eon.s`

Optimization Region (`test_eon.s`)

```
slothy_start:
ldp x2, x3, [x0]
ldp x4, x5, [x0, #16]
ldp x19, x20, [x1]
ldp x21, x22, [x1, #16]

eon x2, x2, x19
eon x3, x3, x20
eon x4, x4, x21
eon x5, x5, x22

stp x2, x3, [x0]
stp x4, x5, [x0, #16]
slothy_end:
```

Note

- Simple test with 4 EON operations
- You don't need to modify the assembly

Assignment 4: `optimize.py`

`optimize.py`

```python
# Initialize SLOTHY
slothy = Slothy(AArch64_Neon, Target_CortexA55)

# Load the test assembly file
slothy.load_source_from_file("test_eon.s")

# Optimize between markers
slothy.optimize(start="slothy_start", end="slothy_end")

# Write optimized output
slothy.write_source_to_file("test_eon_optimized.s")

```

Note: You do not need to modify `optimize.py` - it will work once EON is added

Assignment 4: Summary & Bonus Challenge

Your Two Steps

1. **Architecture Model:** Add EON parsing to `aarch64_neon.py`
2. **Microarchitecture Model:** Add EON performance data to `cortex_a55.py`

Assignment 4: Summary & Bonus Challenge

Your Two Steps

1. **Architecture Model:** Add EON parsing to `aarch64_neon.py`
2. **Microarchitecture Model:** Add EON performance data to `cortex_a55.py`

Bonus Challenge - Prize Available!

- First person at CHES to open a PR adding EON to upstream SLOTHY wins a bottle of Taiwanese Whisky!
- Requirements:
  - Must pass CI
  - Must support Cortex-A72 and Neoverse N1 models too
  - Add EON instruction to `tests/naive/aarch64/instructions.s` for CI testing

Assignment 4: Summary & Bonus Challenge

Your Two Steps

1. **Architecture Model:** Add EON parsing to `aarch64_neon.py`
2. **Microarchitecture Model:** Add EON performance data to `cortex_a55.py`

Bonus Challenge - Prize Available!

- First person at CHES to open a PR adding EON to upstream SLOTHY wins a bottle of Taiwanese Whisky!
- Requirements:
  - Must pass CI
  - Must support Cortex-A72 and Neoverse N1 models too
  - Add EON instruction to `tests/naive/aarch64/instructions.s` for CI testing

Other bonus assignments

- Also support w-form `EON`: `eon w0, w1, w2`
- Also support Barrel-shifted `EON`: `eon x0, x1, x2, lsl #8`

Assignment 5: Instruction Fusion

Input Code: `fusion.s`

```
// Computing column parity in Keccak theta step
start:
    eor v10.16b, v0.16b, v5.16b     // C[0] = A[0,0] ^ A[1,0]
    eor v10.16b, v10.16b, v11.16b   // C[0] = C[0] ^ A[2,0]
    eor v10.16b, v10.16b, v16.16b   // C[0] = C[0] ^ A[3,0]
    eor v10.16b, v10.16b, v21.16b   // C[0] = C[0] ^ A[4,0]
end:
```

Dependencies to Notice

- Chain dependency: Each `eor` depends on previous result
- Target: Use `eor3` instruction from AArch64 SHA3 extension

Goal: Write `optimize.py` using SLOTHY's fusion capabilities

Fusion API Functions

`slothy.fusion_region(start, end, ssa=True)`

Apply fusion callbacks to straightline code region.

- **Input:** `start`, `end` - region labels
- **Output:** Transforms code in-place using fusion callbacks
- **Note:** `ssa`: Output is in static single-assignment (SSA) form (symbolic registers) - usually useful as fusion is used as a pre-processing step before optimization
- **Usage:** `slothy.fusion_region("start", "end", ssa=False)`

Fusion API Functions

`slothy.fusion_region(start, end, ssa=True)`

Apply fusion callbacks to straightline code region.

- **Input:** `start`, `end` - region labels
- **Output:** Transforms code in-place using fusion callbacks
- **Note:** `ssa`: Output is in static single-assignment (SSA) form (symbolic registers) - usually useful as fusion is used as a pre-processing step before optimization
- **Usage:** `slothy.fusion_region("start", "end", ssa=False)`

`slothy.fusion_loop(loop_lbl, ssa=True)`

Apply fusion callbacks to loop body.

- **Input:** `loop_lbl` - loop label
- **Output:** Transforms loop body using fusion callbacks

Hands-on Exercise

[40 minutes hands-on exercise (Assignment 4 + 5)]

See the `README.md` for hints

Tutorial Assignments
github.com/dop-amin/ches2025-slothy-tutorial

Tutorial Slides
kannwischer.eu/talks/20250914_slothy.pdf

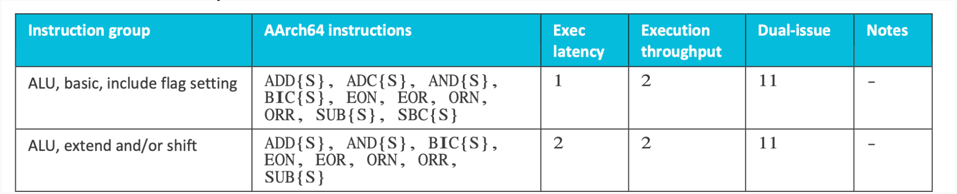
Assignment 4: Solution - Architecture Model

Step 1: Add EON to `aarch64_neon.py`
Add this class (e.g., after the existing `eor` class):

```python
class eon(AArch64Instruction):
    pattern = "eon <Xd>, <Xa>, <Xb>"
    inputs = ["Xa", "Xb"]
    outputs = ["Xd"]
```

Assignment 4: Performance Data from ARM SWOG

Cortex-A55 Software Optimization Guide



Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ALU, basic, include flag setting	ADD{S}, ADC{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}, SBC{S}	1	2	11	-
ALU, extend and/or shift	ADD{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}	2	2	11	-

- Basic EON: Latency = 1, Throughput = 2 (2 ALUs, each inverse throughput = 1)
- Shifted EON: Latency = 2, Throughput = 2
- Dual-issue = 11 (both issue slots = can dual issue)

Assignment 4: Solution - Microarchitecture Model

Add EON to `cortex_a55.py` in 3 dictionaries

```
# 1. Execution Unit
execution_units = {
    eor: ExecutionUnit.SCALAR(),
    eon: ExecutionUnit.SCALAR(), # <- Add this - ALU0 or ALU1
    # ExecutionUnit.SCALAR() short for [ExecutionUnit.ALU0, ExecutionUnit.ALU1]
    ...
# 2. Throughput (per execution unit!)
inverse_throughput = {
    eor: 1,
    eon: 1,  # <- Add this
    ...
# 3. Latency
latencies = {
    eor: 1,
    eon: 1,  # <- Add this
    ...
```

Assignment 4: Alternative Solution Using Class Hierarchy

Architecture Model - Using AArch64Logical Base Class

```python
# Option 1: Direct inheritance (shown earlier)
class eon(AArch64Instruction):
    pattern = "eon <Xd>, <Xa>, <Xb>"
    inputs = ["Xa", "Xb"]
    outputs = ["Xd"]

# Option 2: Using class hierarchy (more elegant)
class eon(AArch64Logical):
    pattern = "eon <Xd>, <Xa>, <Xb>"
    inputs = ["Xa", "Xb"]
    outputs = ["Xd"]

```

Benefit: Only need one entry in Microarchitecture Model for all instructions

Assignment 5: Solution

EOR3 Fusion Solution

```python
from slothy import Slothy
import slothy.targets.aarch64.aarch64_neon as AArch64_Neon
import slothy.targets.aarch64.cortex_a55 as Target_CortexA55

def main():
    slothy = Slothy(AArch64_Neon, Target_CortexA55)

    # Load the source assembly file
    slothy.load_source_from_file("fusion.s")

    # Configure optimization parameters
    slothy.config.outputs = ["v10"]

    # Perform fusion
    slothy.fusion_region(start="start", end="end", ssa=False)

    # Write optimized code to output file
    slothy.write_source_to_file("fusion_opt_a55.s")
```

Common problem 1: Missing instructions from architecture model

Error Message

```
ERROR:root:Failed to parse instruction ldr q0, [x1, #16]!
ERROR:root:A list of attempted parsers and their exceptions follows.
...
  File "slothy/targets/aarch64/aarch64_neon.py", line 792, in parser
    raise Instruction.ParsingException(
slothy.targets.aarch64.aarch64_neon.Instruction.ParsingException:
Couldn't parse ldr q0, [x1, #16]!
You may need to add support for a new instruction (variant)?
```

Solution

- Check if instruction variant is supported in architecture model
- Add new instruction parser to **`aarch64_neon.py`** if needed

SLOTHY models are built lazily - it is expected that not all instructions are supported. AArch64 coverage is fairly good by now.

Common problem 2: Missing instructions from microarchitecture model

Error Message

```
INFO:slothy.slothy_start.slothy:Attempt optimization with max 0 stalls...
Traceback (most recent call last):
...
  File "slothy/targets/aarch64/cortex_a55.py", line 639, in get_inverse_throughput
    return lookup_multidict(inverse_throughput, src)
  File "slothy/targets/aarch64/aarch64_neon.py", line 4815, in lookup_multidict
    raise UnknownInstruction(f"Couldn't find {instclass} for {inst}")
slothy.targets.common.UnknownInstruction:
Couldn't find <class 'slothy.targets.aarch64.aarch64_neon.vmls'> for mls v11.4S, v12.4S, v13.4S
```

Solution

- Instruction is parsed correctly but missing performance data in microarchitecture model
- Add execution unit assignment and latency data to target model (e.g., `cortex_a55.py`)
- Check software optimization guide for the target microarchitecture

SLOTHY models are built incrementally - not all instructions have complete microarchitecture data yet.

Common problem 3: Trying to optimize non-straight-line code

Code

```
start:
  ldr q0, [x1], #16
  add v1.4s, v0.4s, v0.4s
  str q1, [x2], #16

  subs x3, x3, #1
  b.ne start  // <-- Problem!

  mov v2.16b, v1.16b
end:
```

Trying to use: `slothy.optimize(start="start", end="end")`

Error Message

```
ERROR:root:Failed to parse instruction b.ne start
...
slothy.targets.aarch64.aarch64_neon
  .Instruction.ParsingException:
Couldn't parse b.ne start
You may need to add support for a new
instruction (variant)?
```

Solution

- SLOTHY cannot optimize across branches
- For loops: use `optimize_loop("start")`
- For other branches: optimize each basic block separately
- Place markers to avoid branch instructions

Common problem 4: Useless instruction warnings

Loop Code

```
loop_start:
  ldr q0, [x0], #16
  ldr q1, [x1], #16

  add v2.4s, v0.4s, v1.4s

  str q2, [x2], #16

  subs x4, x4, #1
  b.ne loop_start
```

Error Message

```
ERROR:slothy.loop_start.slothy.dataflow:
The result registers ['x0'] of instruction
0:[ldr q0, [x0], #16] are neither used nor
declared as global outputs.
ERROR:slothy...dataflow:This is often a configuration
  error. Did you miss an output declaration?
  ...
slothy.core.dataflow.SlothyUselessInstructionException:
Useless instruction detected
```

Solution

- Use: **`config.inputs_are_outputs = True`** for loops
- Use: **`config.outputs = [...] for other outputs`**
- Warnings: **`config.allow_useless_instructions`** should not be used in those cases
- Note: Can also be caused by incorrect input/output declarations in the arch. model

Common problem 5: SLOTHY using callee-saved registers

Original Function

```
.global vec_add
vec_add:
start:
  ldr q0, [x0]
  ldr q1, [x1]
  add v2.4s, v0.4s, v1.4s
  str q2, [x0]
end:
  ret
```

SLOTHY Output (BROKEN!)

```
.global vec_add
vec_add:
start:
  ldr q7, [x0]
  ldr q3, [x1]
  add v8.4s, v7.4s, v3.4s
  str q8, [x0]
end:
  ret
```

Problem

- SLOTHY renamed v2 → v8
- v8-v15 are callee-saved
- Function corrupts v8

Solution

```
# Reserve callee-saved vector regs
slothy.config.reserved_regs = [
  "v8", "v9", "v10", "v11",
  "v12", "v13", "v14", "v15"
]
```

Thank You!

Thank you for joining the SLOTHY tutorial!

Get Involved:

- Found a bug or have a feature request? Open an issue on GitHub!
- Want to contribute? Pull requests are always welcome!
- We have plenty of ideas for further research
- Questions? Feel free to reach out directly

Contact:
Matthias Kannwischer matthias@kannwischer.eu
Amin Abdulrahman amin.abdulrahman@mpi-sp.org

GitHub: github.com/slothy-optimizer/slothy

Join our Discord!
discord.gg/Khy2bwgm