Optimized Cryptography with LLVM Non-Standard Bitwidth Types

Michael McLoughlin
mcloughlin@cmu.edu

https://mmcloughlin.com/15745

Non-standard Bitwidth Types

Compiler optimization for domain-specific use cases

Background

LLVM has arbitrary bitwidth types. C23 will soon have `_BitInt(N)`. *What can we do with them?*

Opportunity

Bitwidth types allow concise and safer implementations of otherwise complex cryptography. *Can compiler optimizations reach parity with hand-tuned code?*

Contribution

Demonstrate feasiblity of finite-field cryptography with bitwidth types: achieved optimizations of 80.1x over baseline getting within 1.6x of hand-tuned code.

Experimental Setup

Target benchmark: X25519 elliptic curve with `_BitInt(255)` type

X25519 cryptographic algorithm: Widely deployed and *representative* of finite-field crypto.

- Base operations arithmetic modulo prime $2^{255} - 19$. Implemented with C type `_BitInt(255)`.
- Inner loop is modular arithmetic expression graph (right)

Baseline: Stock Clang/LLVM is 127.3x slower than `libsodium` hand-tuned code.

```
A = X2+Z2
AA = A2
B = X2-Z2
BB = B2
E = AA-BB
C = X3+Z3
D = X3-Z3
DA = D*A
CB = C*B
X5 = (DA+CB)2
Z5 = X1*(DA-CB)2
X4 = AA*BB
Z4 = E*(BB+a24*E)
```

Crandall Reduction

Fast modular reduction trick for special primes

$x \mod p$ optimization for *Crandall primes* of the form:

$$p = 2^n - c \implies 2^n \equiv c \pmod{p}$$

Crandall reduction step folds the high-bits into the low n bits: Output much smaller than x and equivalent modulo p.

$x = 2^n H + L =$

H	L (n bits)

$$cH + L$$

Generate Crandall steps prior to `urem` instructions.

CrandallReductionPass: 24.0x speedup!

Reduction Analysis

Identify expression graphs of pure modular arithmetic

Goal: identify arithmetic values that *will be reduced modulo p*.

$$\begin{tikzcd}
x_2 \arrow[dr] & & z_2 \arrow[dl] & & x_3 \arrow[dr] & & z_3 \arrow[dl] \\
& D = (x_2 + z_2) \mod p \arrow[drr] & & & & A = (x_3 - z_3) \mod p \arrow[dll] & \\
& & & (D * A) \mod p & & &
\end{tikzcd}$$

`ReductionAnalysis` pass. Worklist algorithm:

1. Value lattice states: `UNDEF`, `REDUCED(M)`, `NOTREDUCED`
2. Reduction *generated* by `urem` instructions
3. Propagated: *non-wrapping* arithmetic, no-op casts, ϕ

Incomplete Reduction

Eliminate expensive reductions by operating in partially reduced form

Crandall steps leave a partially reduced result. Can we stop there? *Yes*, if Reduction Analysis tells us the value will be reduced in future.

Algorithm

Goal: replace `urem` instructions with Crandall reductions, resize types as necessary.

1. Plan: which `urem` reductions will be left incomplete
2. Type resizing: what types must the new graph have?
3. Rewrite: rewrite modular arithmetic expression graph

Incomplete Reduction: Extra 2.1x. 3.3x with ϕ handling.

Results

Combined Results: *80.1x speedup over baseline!* Only 1.6x away from hand-tuned.

Implementation	Time (us)
Baseline	4682.75
Crandall Single Step	195.29
Crandall Multi Step	194.95
Incomplete Reduction	93.11
Incomplete Reduction over Phi	58.46
libsodium	36.80

Conclusion and Future Directions

Conclusions

- Cryptography in `_BitInt(N)` types is *feasible*
- Domain-specific optimizations within 1.6x of hand-tuned

Future Directions

- Mid-end: more tricks, support more modulus types
- Back-end: Intel ADX, AArch64, Peephole
- Real-world concerns: constant-time, verification
- Custom compiler: MLIR dialect?