Team 58 Presentation

Richard Wohlbold
Aljaz Medic
Aleksa Micanovic
Aleks Stepancic

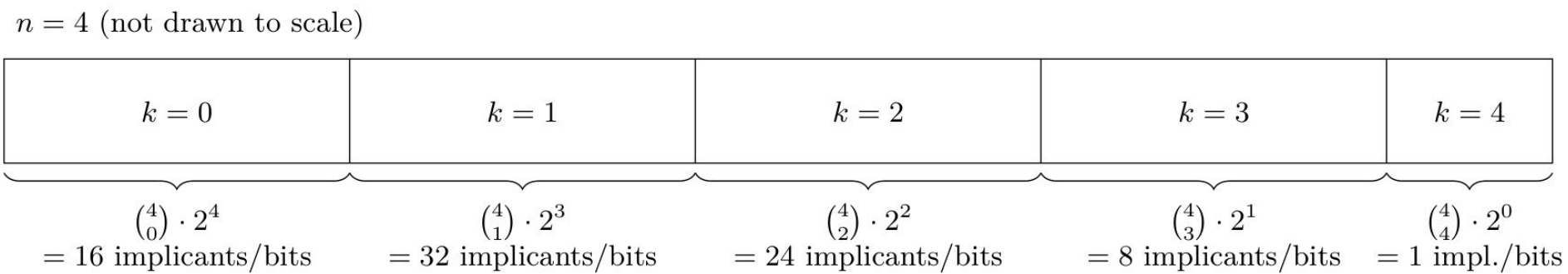
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prime Implicants for Quine-McCluskey

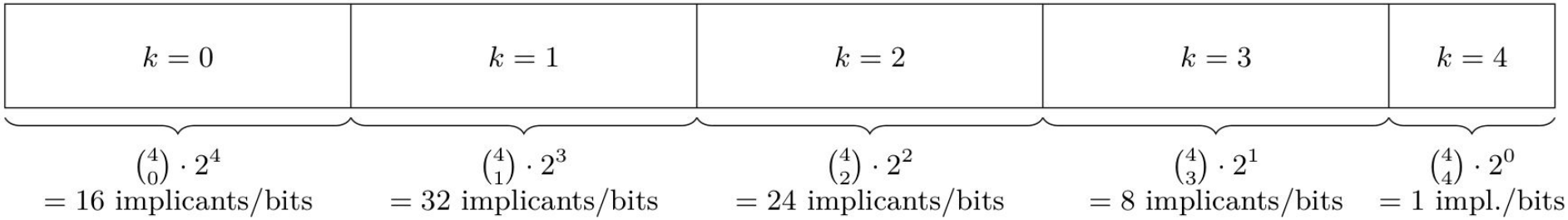
- The algorithm determines all prime implicants of a boolean function, given the initial set of minterms
- **Input:** number of bits and list of minterms
  (e.g. num_bits=7 0010011, 100101, …)
- **Output:** list of all prime implicants

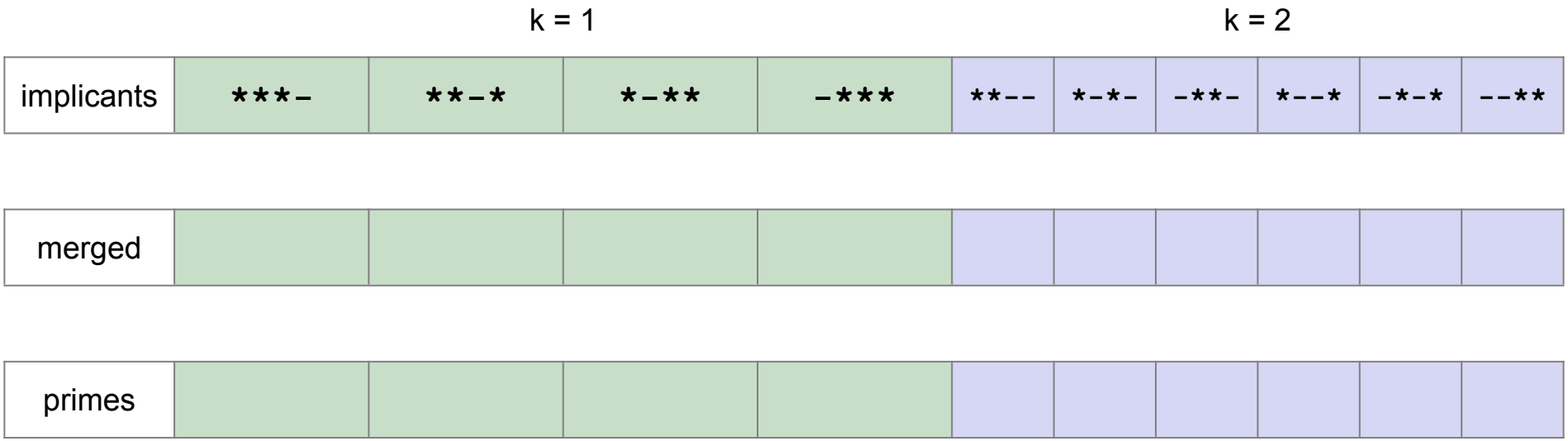
Memory Layout

- **Every implicant has a bit that describes it in a 3^n array**
- **There are 3 such arrays: implicants, merged, primes**
- **Each is divided into n+1 sections made up of chunks**
  - For 0 <= k <= n, the kth section contains:
    - *all implicants with k dashes*
    - *(n choose k) many chunks (all implicants have dashes in the same place)*
    - *chunks of size 2^{n-k} bits*
- **Chunks within the same section have a combinatorial ordering, based on the location of the dashes**
- **A single merge step takes one chunk of input and produces multiple chunks of output**
- **Memory layout achieves very good temporal + spatial locality and is a key reason for our good performance**

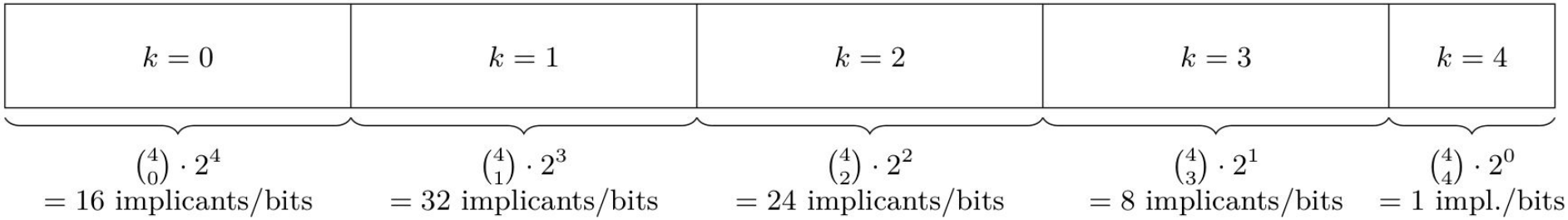


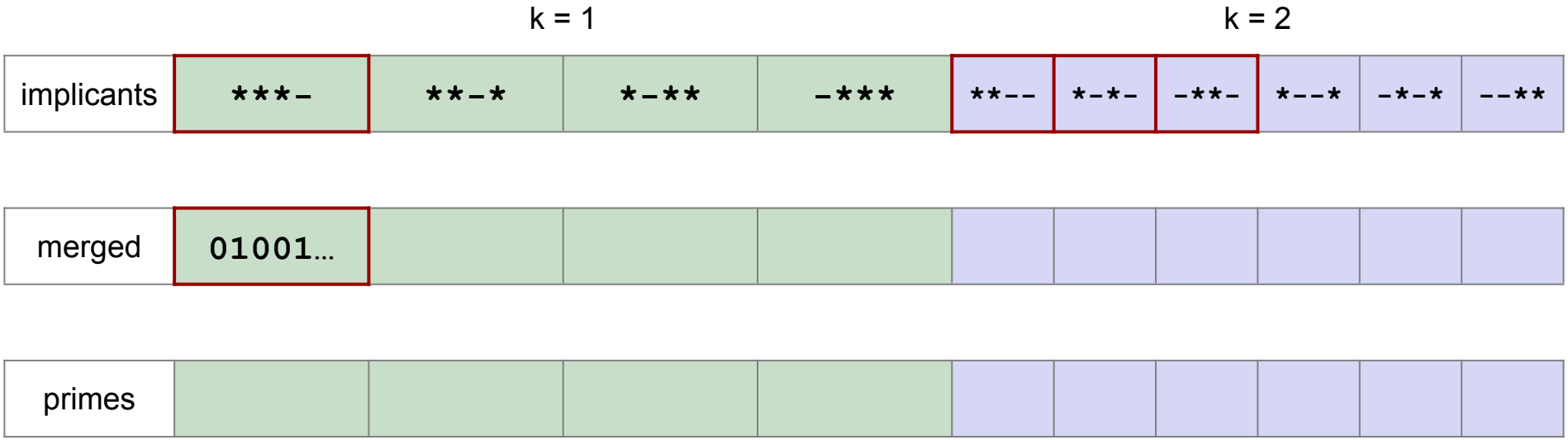
$n = 4$ (not drawn to scale)



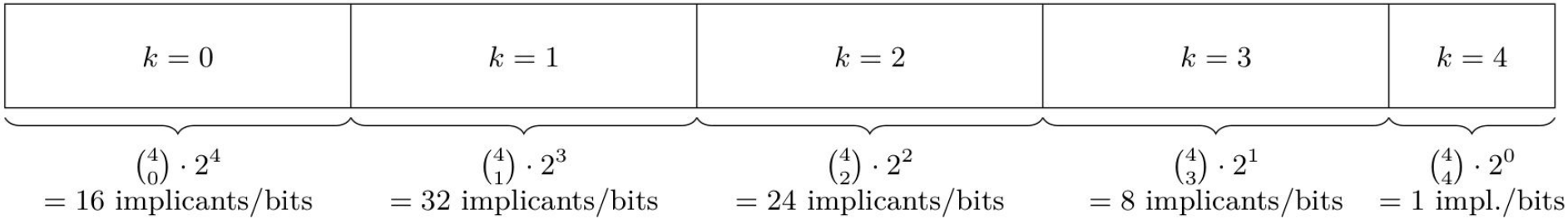


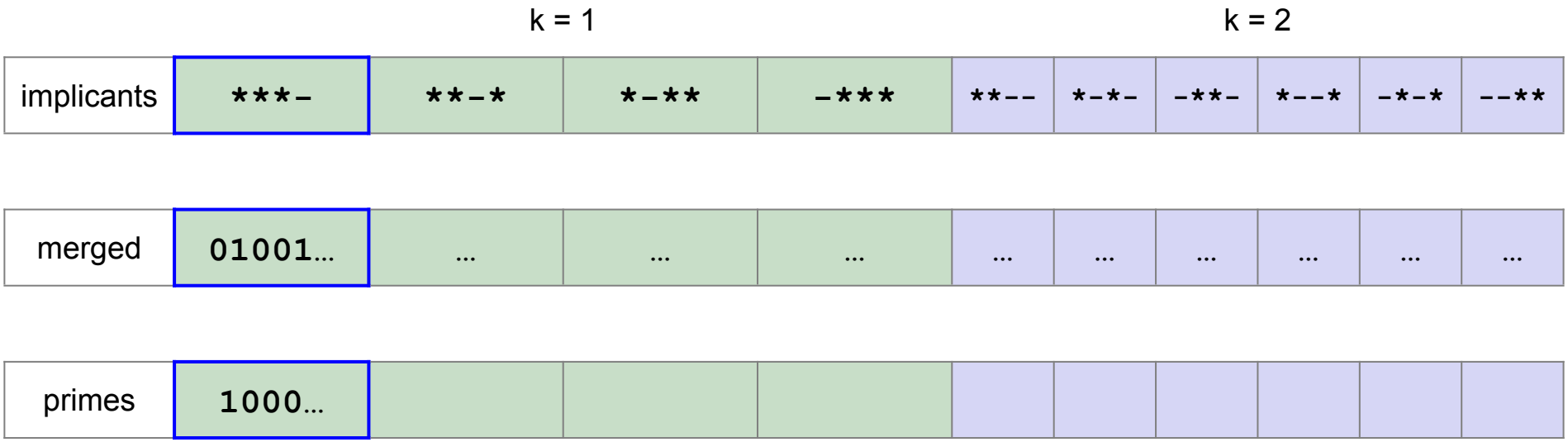
$n = 4$ (not drawn to scale)





$n = 4$ (not drawn to scale)





Cost Analysis

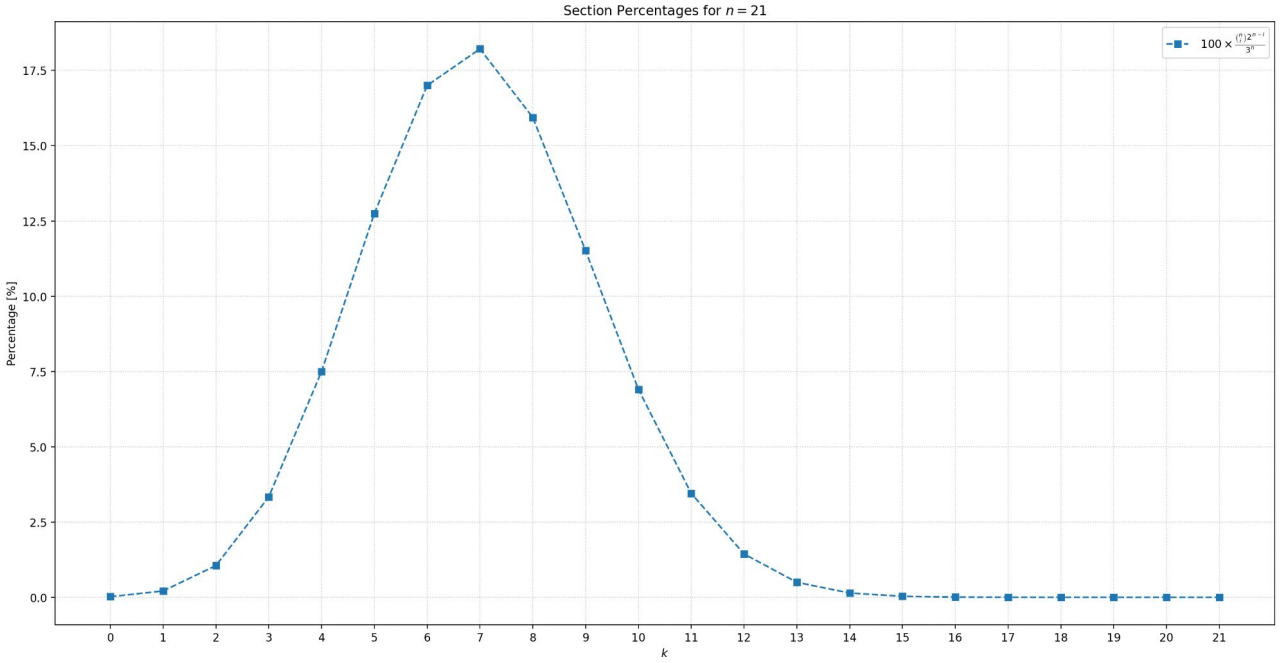
$$\#\text{Bit-Ops} = n3^n$$

$$Q(n) \geq \frac{2 \cdot 3^n}{8} \text{bytes}$$

$$I(n) \leq 4n$$

Takeaway: To optimize performance for certain n, optimize merge performance for num_dashes ≈ n/3

Biggest section for n implicants: $k = \left\lfloor \frac{n+1}{3} \right\rfloor$ dashes



Dummy Baseline “Dense Byte”

- Two phase approach
  1. **Generation (Merge):** AND pair of implicants that differ by one bit
  2. **Reduce:** Find implicants that are not merged
- **Drawback:**

  “Dense byte” stores **one implicant per byte**

Optimization #1 “64bit vectorization”

Improvement:

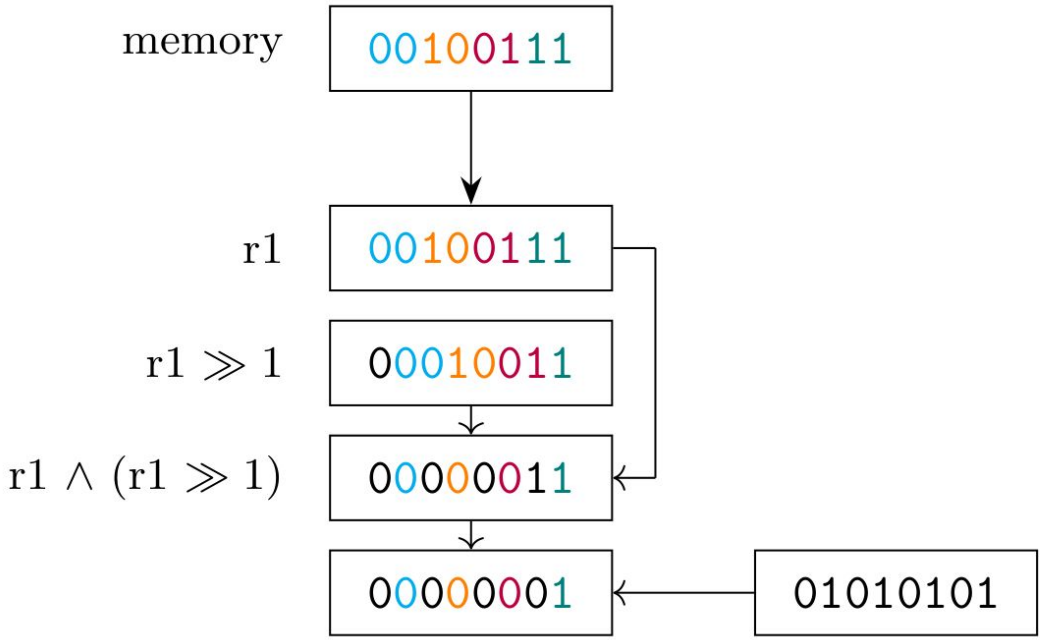
Store 64 implicants in **64 bit registers**

Block sizes >= 64, merging **EASY**

Problem:

Block sizes < 64, need to find approach to extract the merged bits within the register.

$n \geq 3$, block size: 1 bit, register width: 8 bits



How to reorganize bits into lower half, i.e. turn 01010101 into 00001111 ?

Bit Extraction Solution 1: pext

```
unsigned __int64 _pext_u64 (unsigned __int64 a, unsigned __int64 mask)
```

Synopsis

```
unsigned __int64 _pext_u64 (unsigned __int64 a, unsigned __int64 mask)
#include <immintrin.h>
Instruction: pext r64, r64, r64
CPUID Flags: BMI2
```

Description

Extract bits from unsigned 64-bit integer `a` at the corresponding bit locations specified by `mask` to contiguous low bits in `dst`; the remaining upper bits in `dst` are set to zero.

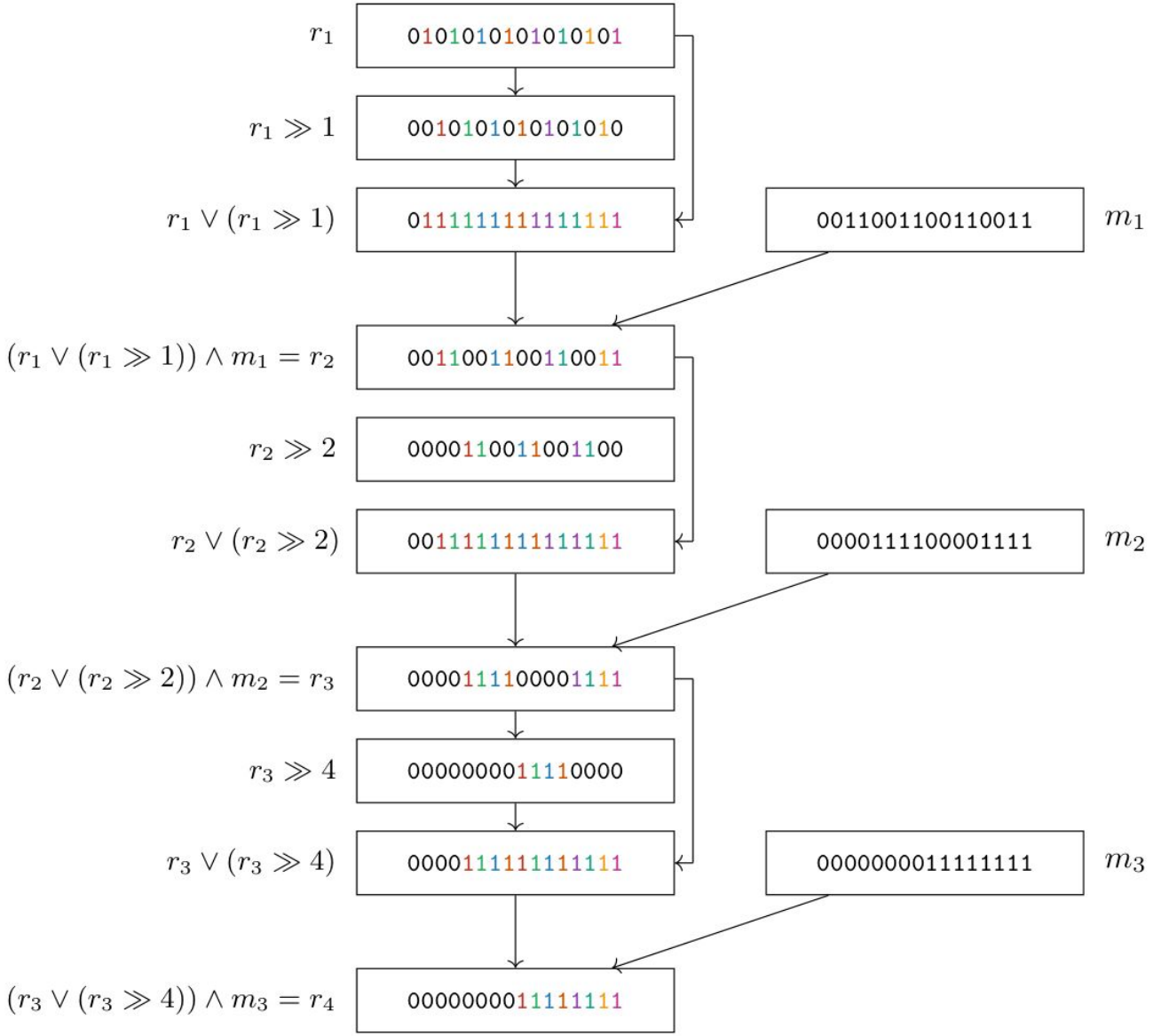
Operation

```
tmp := a
dst := 0
m := 0
k := 0
DO WHILE m < 64
        IF mask[m] == 1
                dst[k] := tmp[m]
                k := k + 1
        FI
        m := m + 1
OD
```

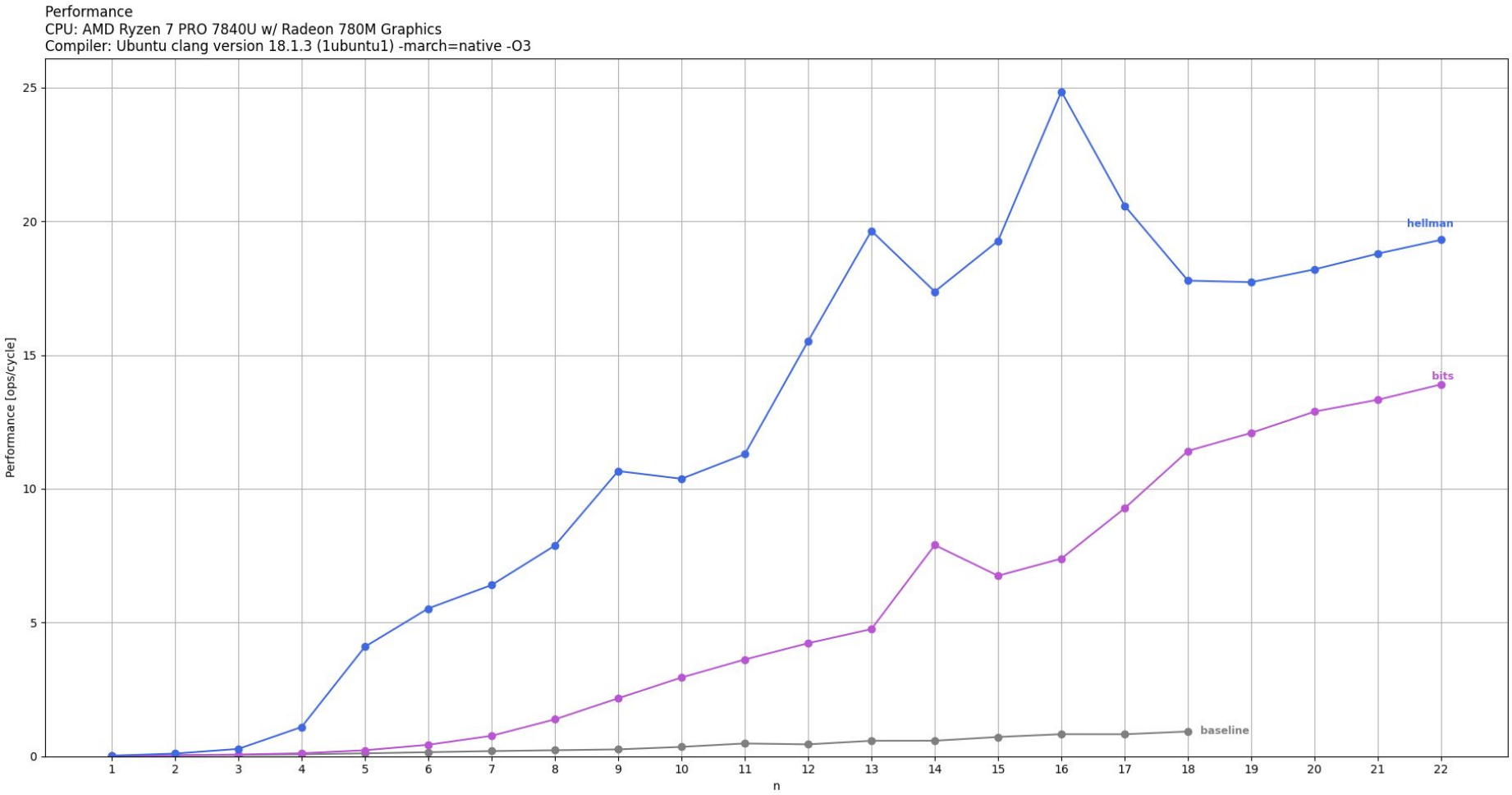
Latency and Throughput

Architecture	Latency	Throughput (CPI)
Alderlake	3	1
Icelake Intel Core	3	1
Icelake Xeon	3	1
Sapphire Rapids	3	1
Skylake	3	1

Bit Extraction Solution 2: Shifting and Masking



PLOT #1 (baseline, bits, hellman)



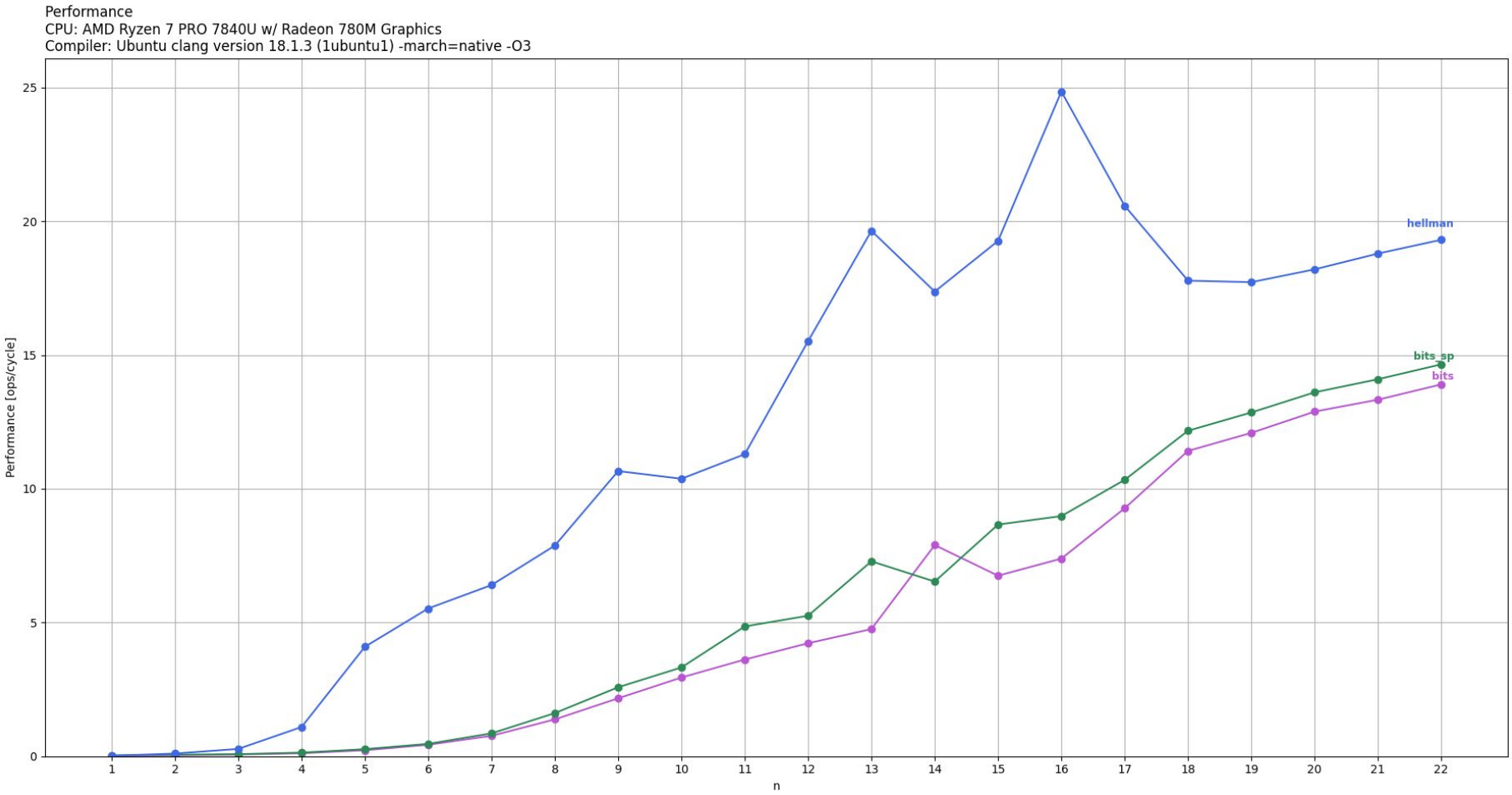
Optimization #2 (single pass)

- **Additional space (3^n bits)** for a merge bitmap
- **Additional time and cache misses** for second prime-marking traversal through **implicants, merge, prime** bitmaps
- **Improvement:** in the first pass, populate **prime** bitmap

```
merged = implicants[i] AND implicants[i + (1 << k)]

primes[i] = primes[i] AND NOT merged
```

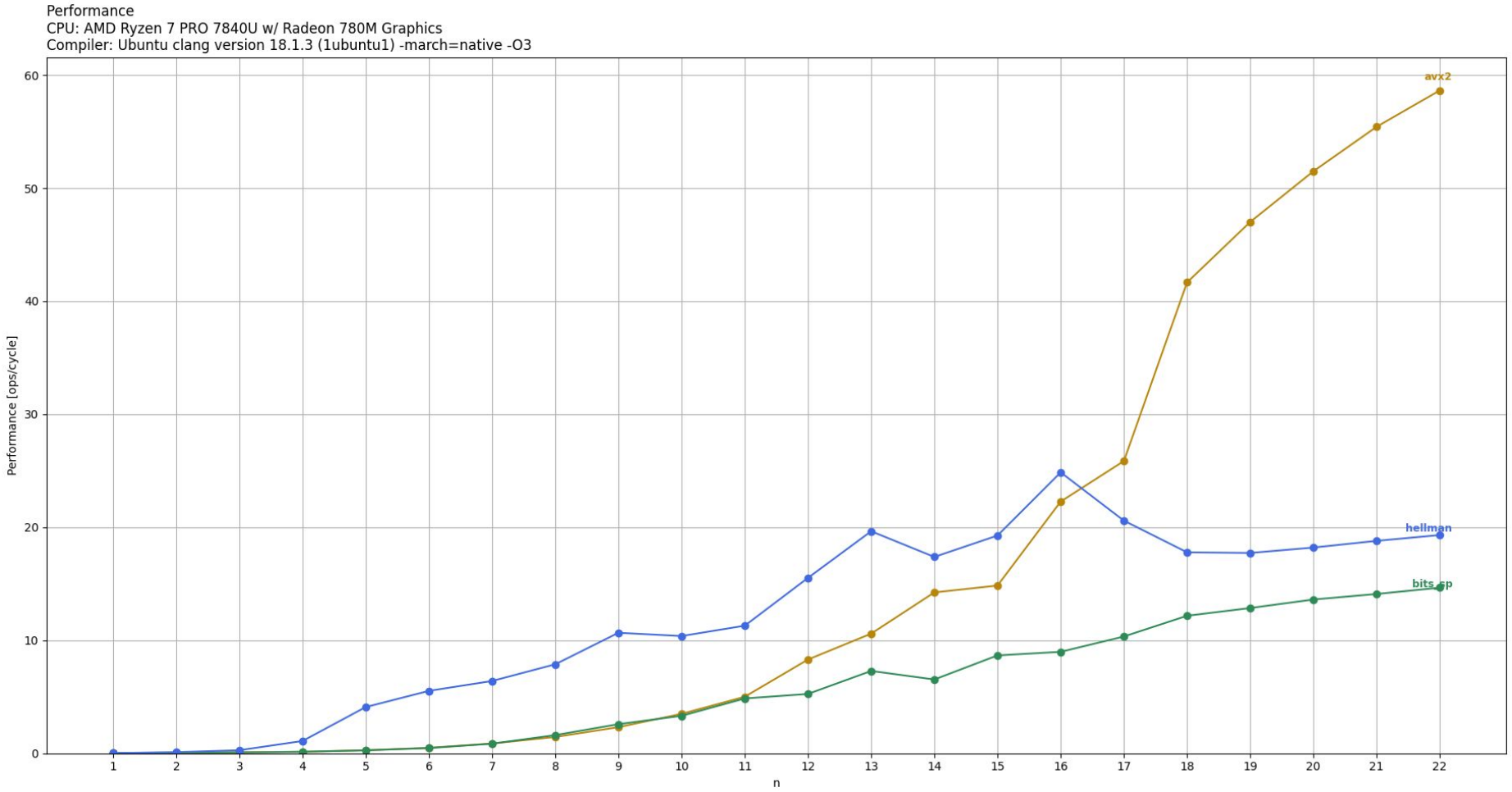
PLOT #2 (bits, bits_sp, hellman)



Optimization #3 (AVX2 Vectorization)

- **Using 64-bit registers**
- **Improvement:** use 256-bit registers

PLOT #3 (avx2, bits_sp, hellman)



N=22 Speedup$_{\text{Avx2 / Hellman}}$ = **3.1**x Speedup$_{\text{Avx2/bits_sp}}$ = **3.9**x

Optimization #4 (SSA+ILP, Unroll)

- **Convoluted IF branches for inter-register merge**
  - No ILP
- **Improvement 1:** write straightforward code - add ILP

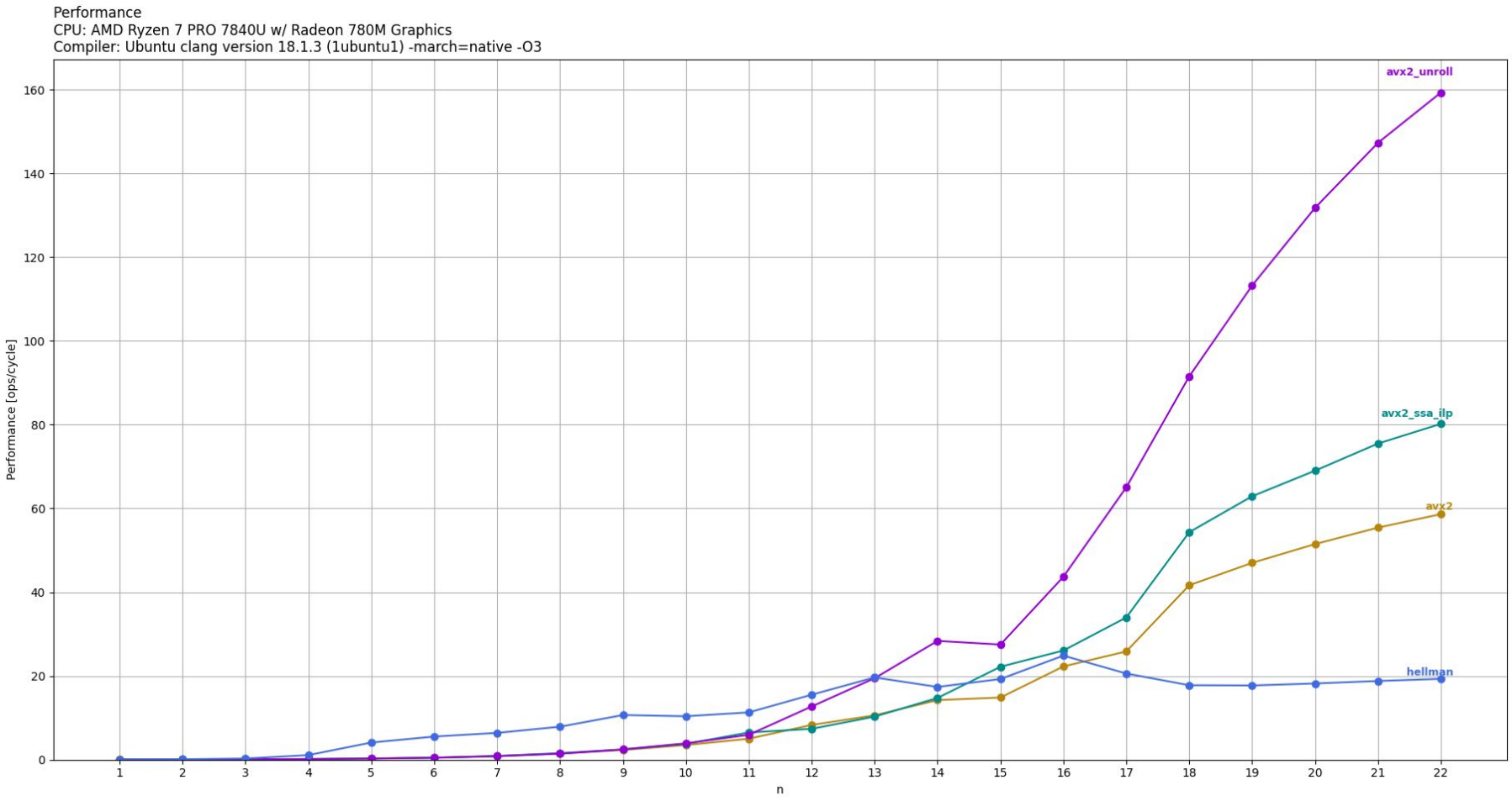
In **merge step** register will be **loaded multiple times** for each block length

$$\#\text{register\_loads} = k\frac{2^k}{\text{register\_size}} = k2^{k-8}$$

- **Suboptimal traversal order of inter-register merge in a chunk**
  - Bad temporal locality - multiple reads of same input bits
  - For large chunks can evict the whole cache
- **Improvement 2:** Inter-register merge processes all **possible block lengths** for a register before moving further + unroll loop for 4 inter-register steps

$$\#\text{register\_loads} = (k-7)2^{k-8}$$

PLOT #4 (avx2, avx2_ssa_ilp, avx2_unroll, hellman)

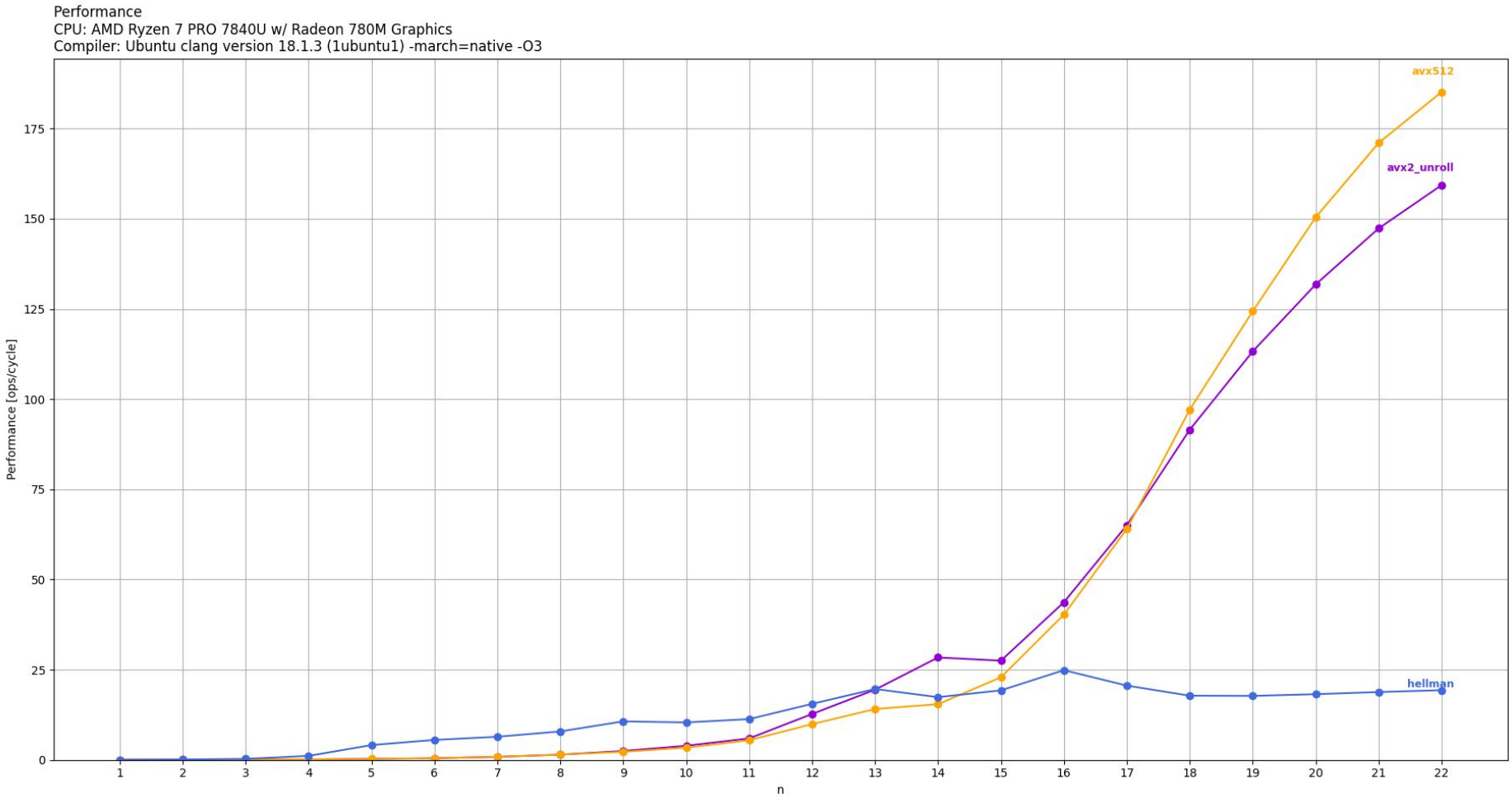


N=22 Speedup$_{\text{avx2_unroll/Hellman}}$ = **8.0**x Speedup$_{\text{avx2_unroll/avx2}}$ = **2.7**x

Optimization #5 (AVX512)

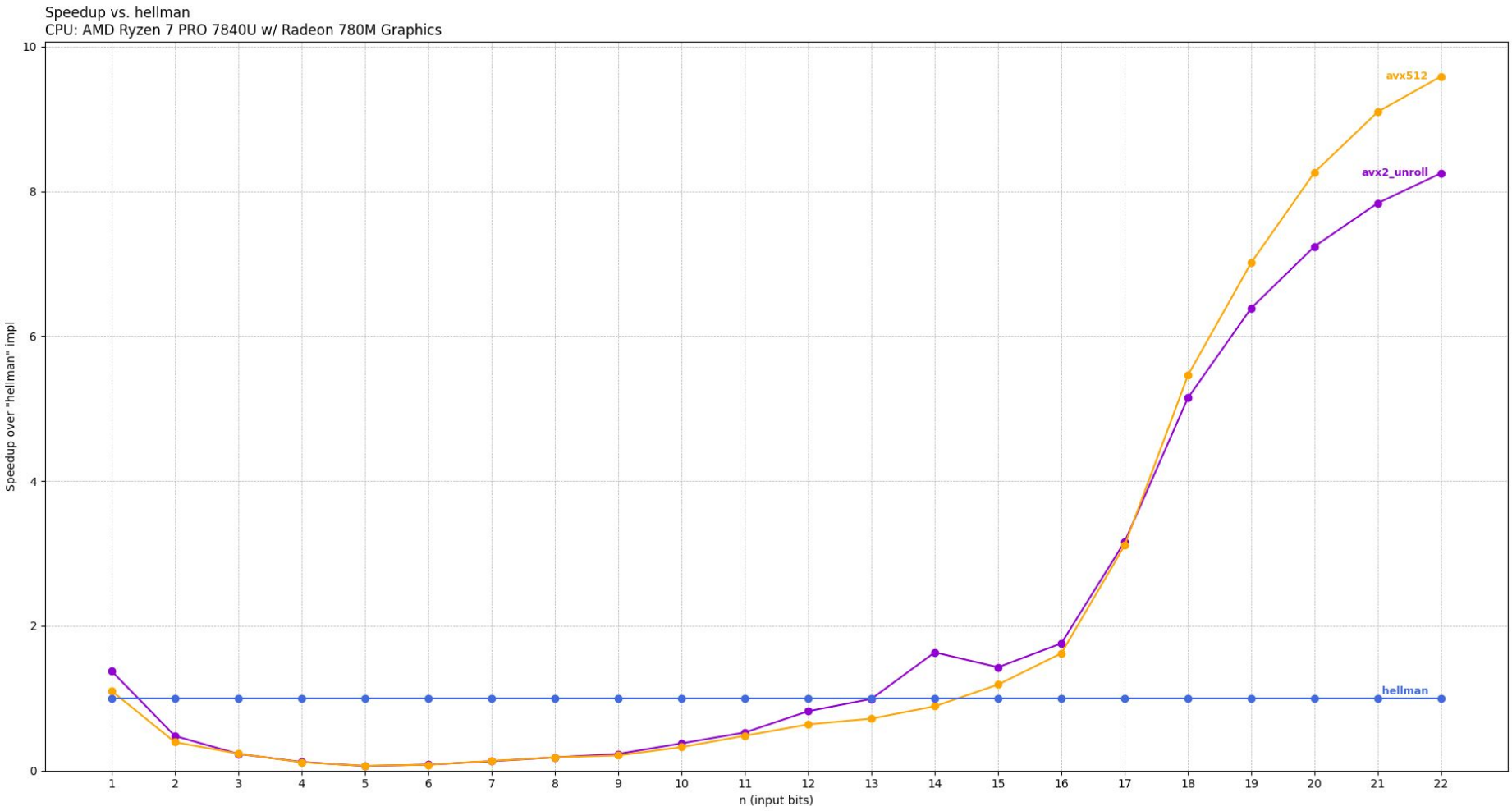
- **Using 256-bit registers**
- **Improvement:** use 512-bit registers

PLOT #5 (avx2_unroll, avx512, Hellman)



N=22 Speedup$_{\text{avx512/Hellman}}$ = **9.3**x Speedup$_{\text{avx512/avx2_sp_unroll}}$ = **1.1**x

SPEEDUP (avx2_unroll, avx512 vs hellman)

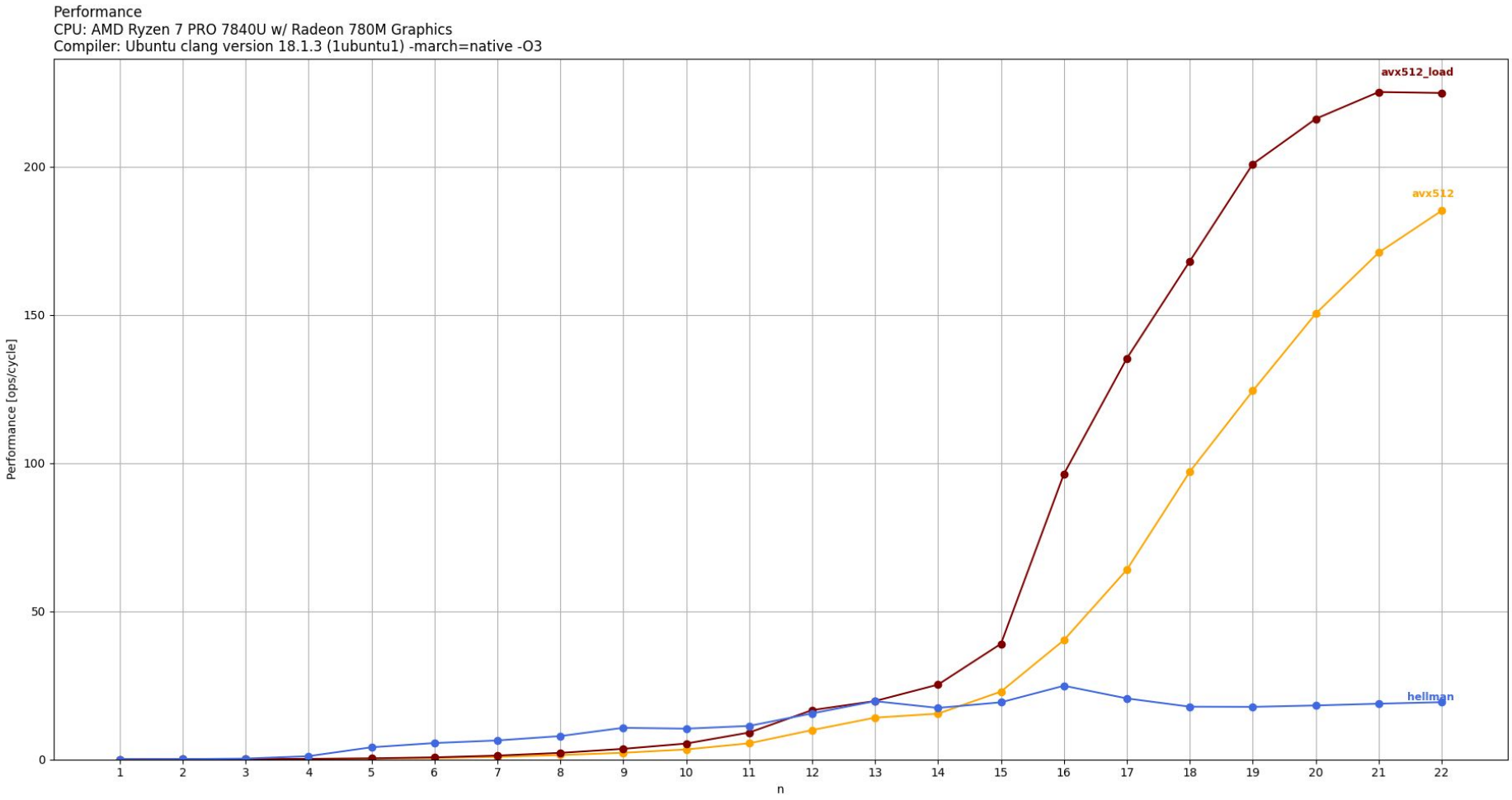


Final Optimization (loading the traversal order)

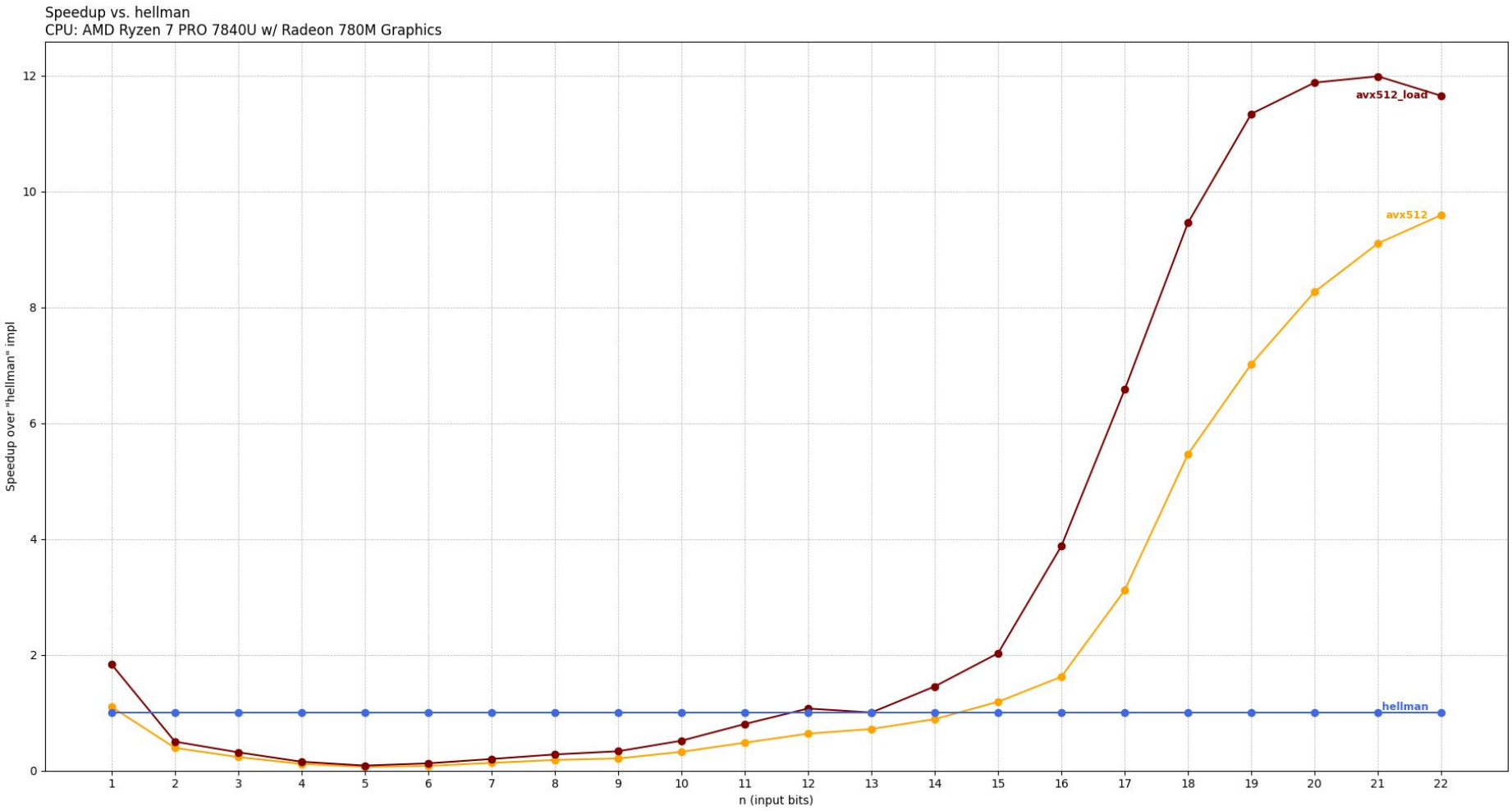
- **The traversal order of implicants is being calculated, which adds cycles to the execution**
- **Improvement:** precompute the traversal order
- **Load additional 18 * 2^n bytes**
  - For n = 20, 21, 22 **negligible compared to bitmaps (< 1% of storage)**

```
// one linear pass
for (size_t i = 0; i < op_count; i++) {
    MergeOp *op = &ops[i];
    MERGE_FUNCTION(
        implicants,
        primes,
        op->in_idx,
        op->out_idx,
        op->rem_bits,
        op->first_diff
    );
}
```

FINAL PLOT #5 (avx512, avx512_load, hellman)

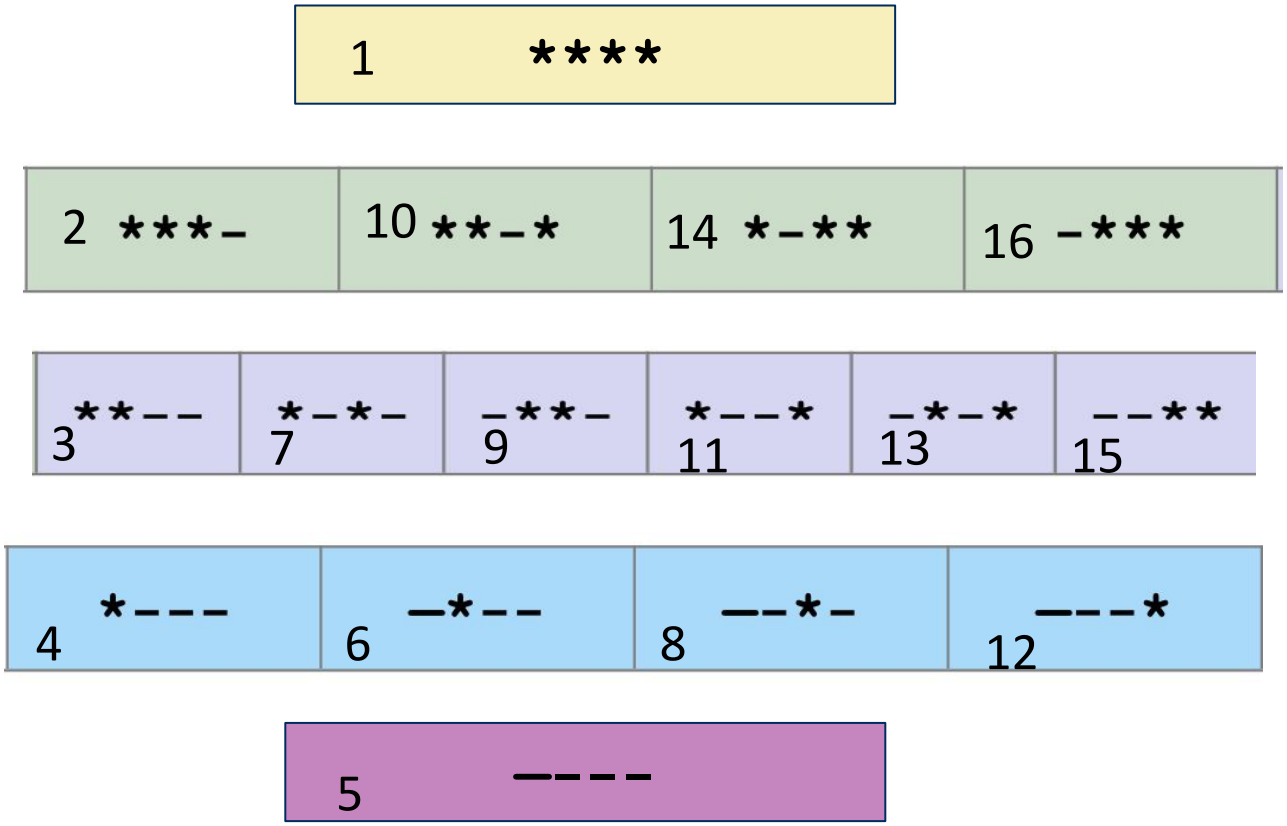


SPEEDUP (avx512, avx512_load vs hellman)

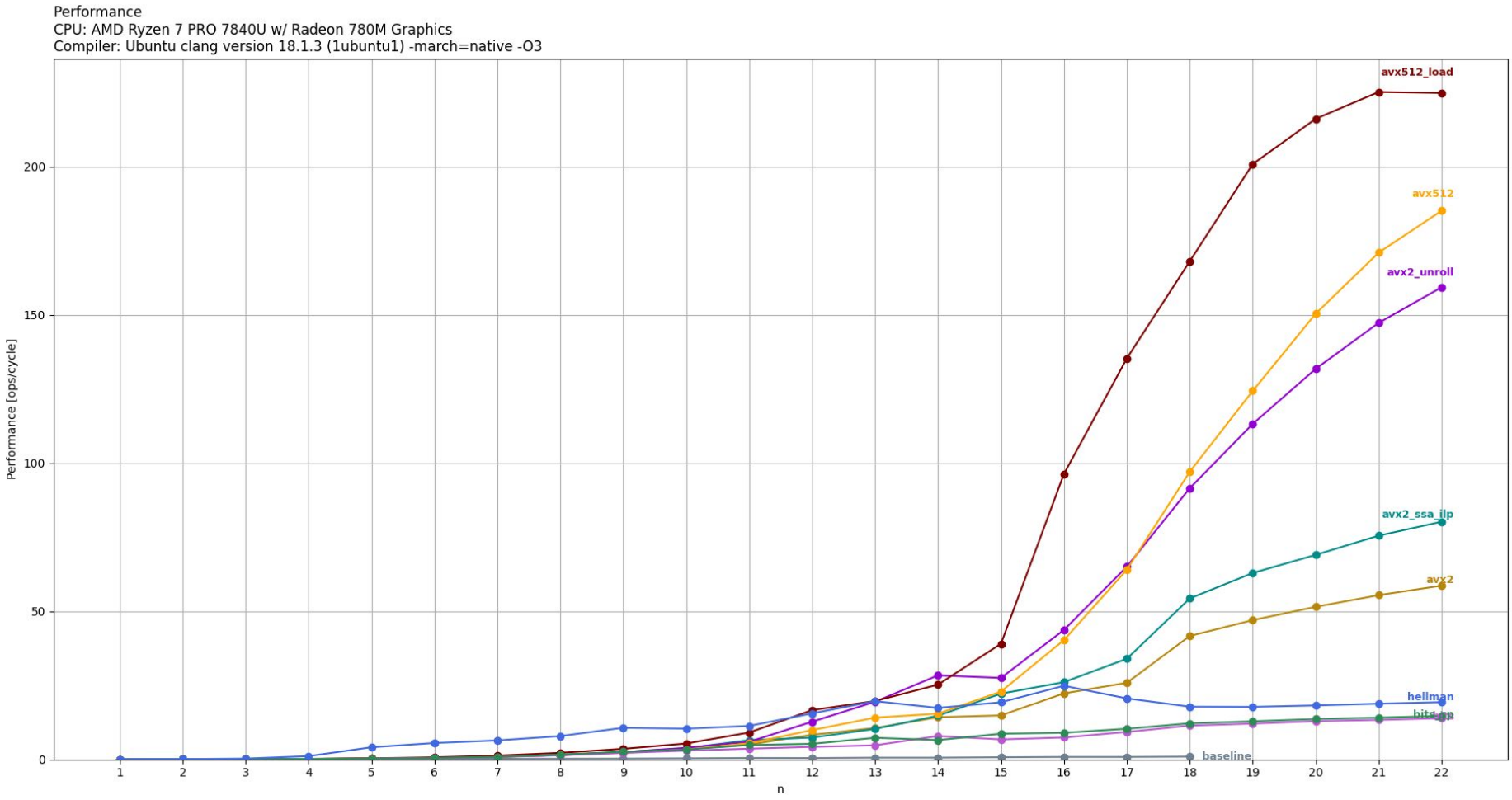


Failed optimization: DFS traversal

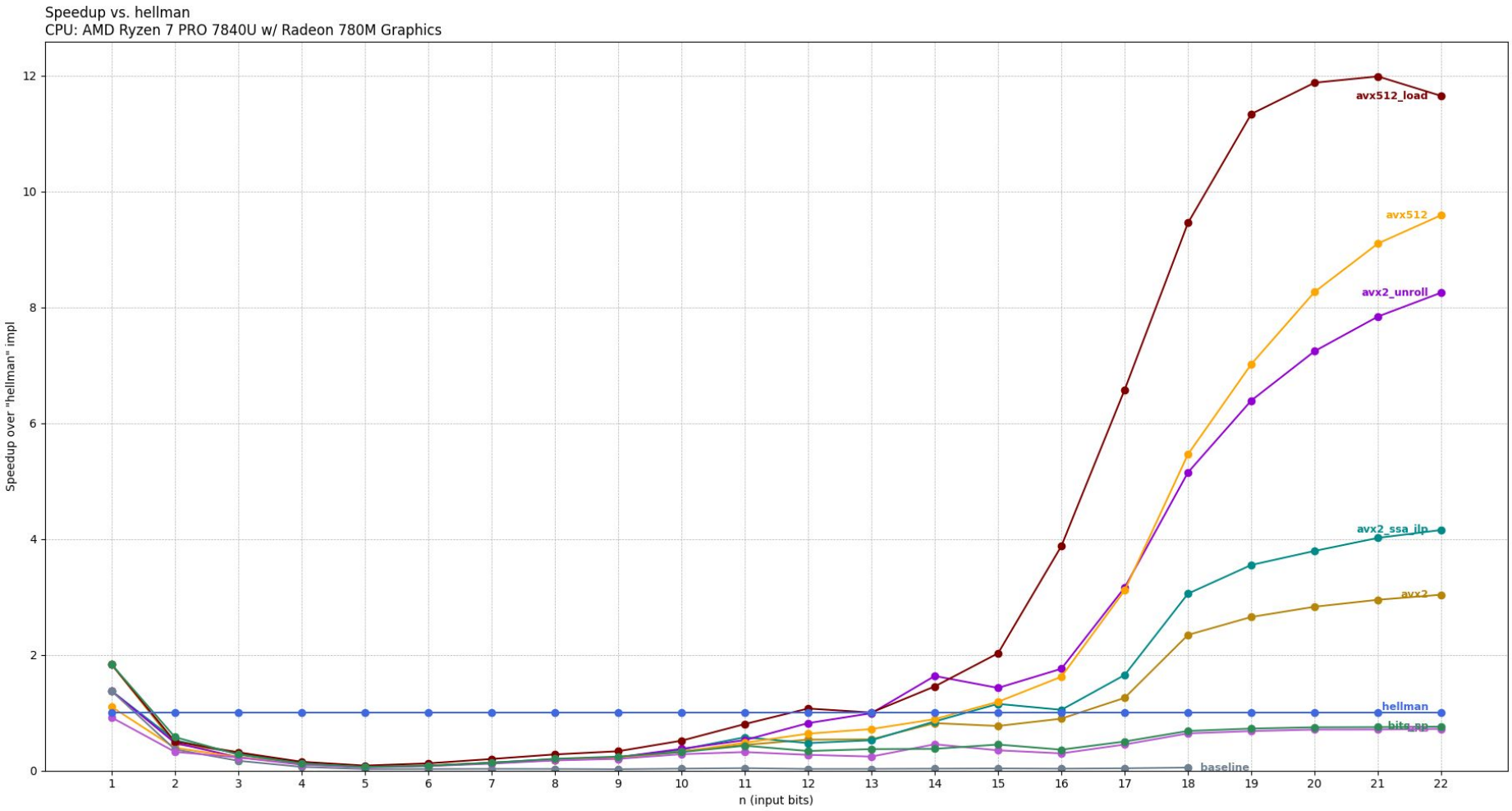
No effect



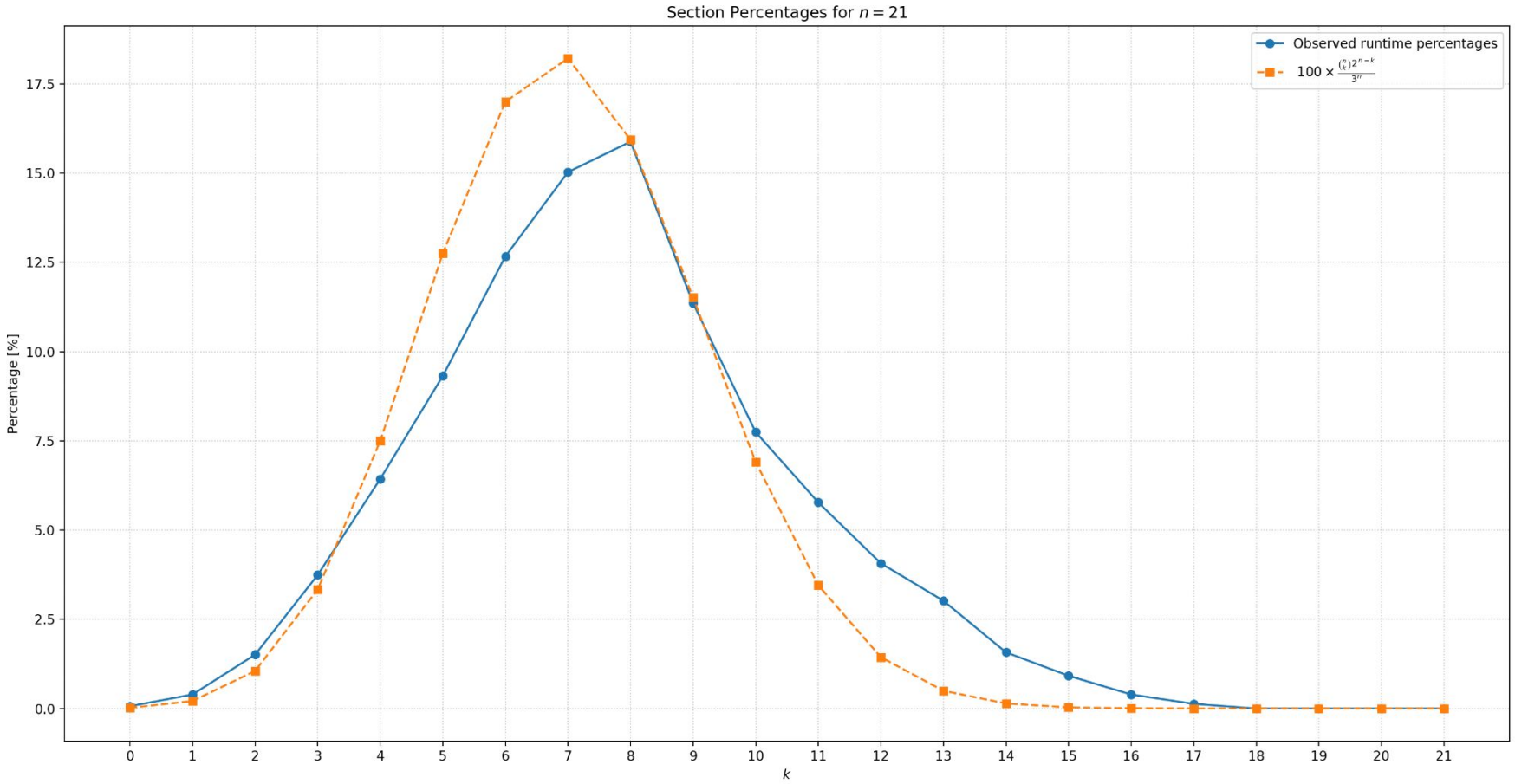
PERF PLOT TOTAL



SPEEDUP PLOT TOTAL



AVX2 SP BLOCK N=21



(few dashes, big blocks) (many dashes, small blocks)

Takeaway: In the theoretical model, for a given merge step, all block sizes take an equal amount of time. On our implementation, small block sizes are slower than larger ones.

Future Optimizations

- **Introduce another layer of blocking**
  - Instead of loading the entire chunk for block sizes >= 256, two registers at a time, load 4, 8, 16, … registers and compute multiple blocks at the same time
- **Further improve bit extraction**
  - Example: AVX512 GFNI instructions

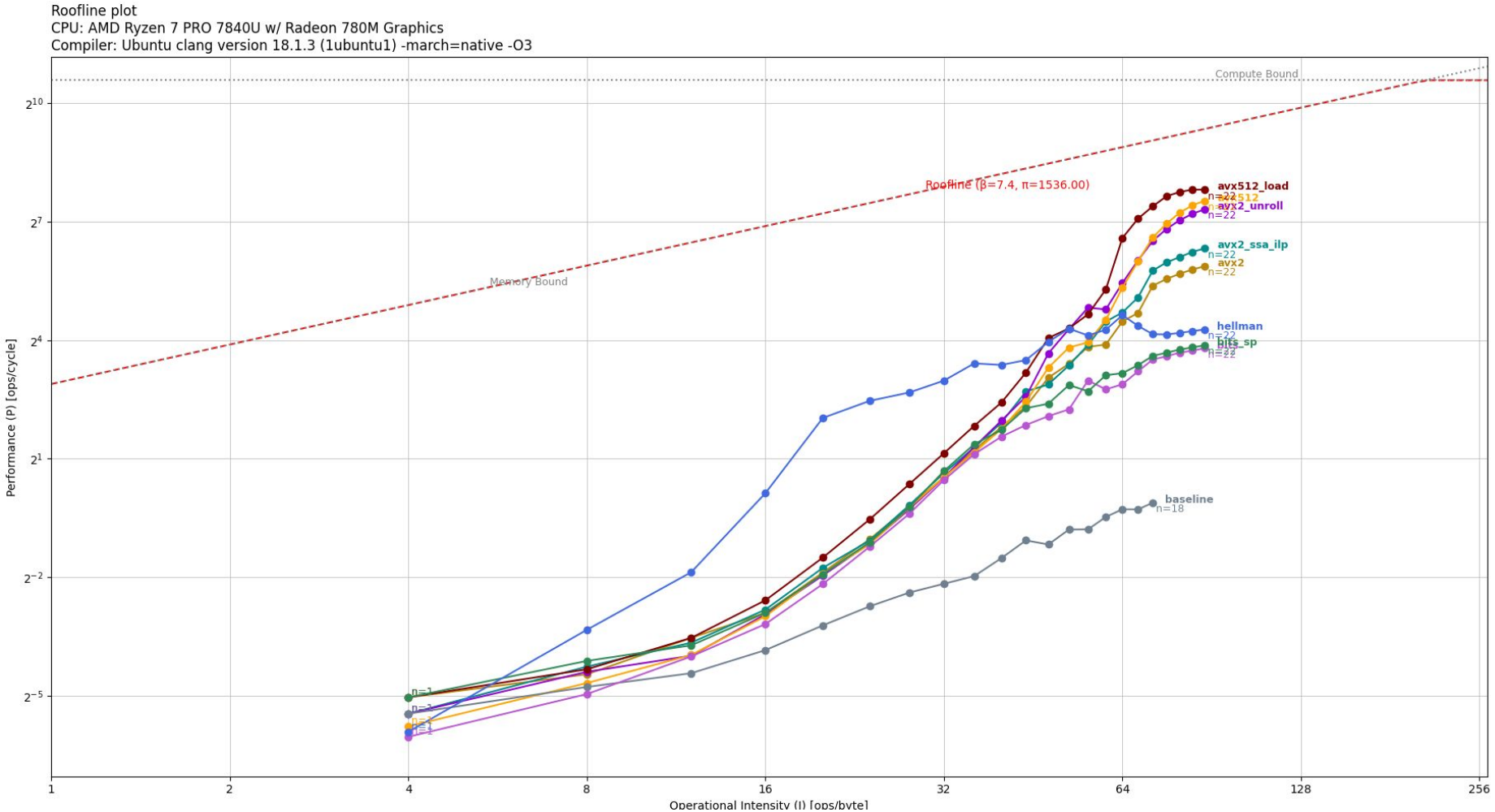
Thank you!

Team 58

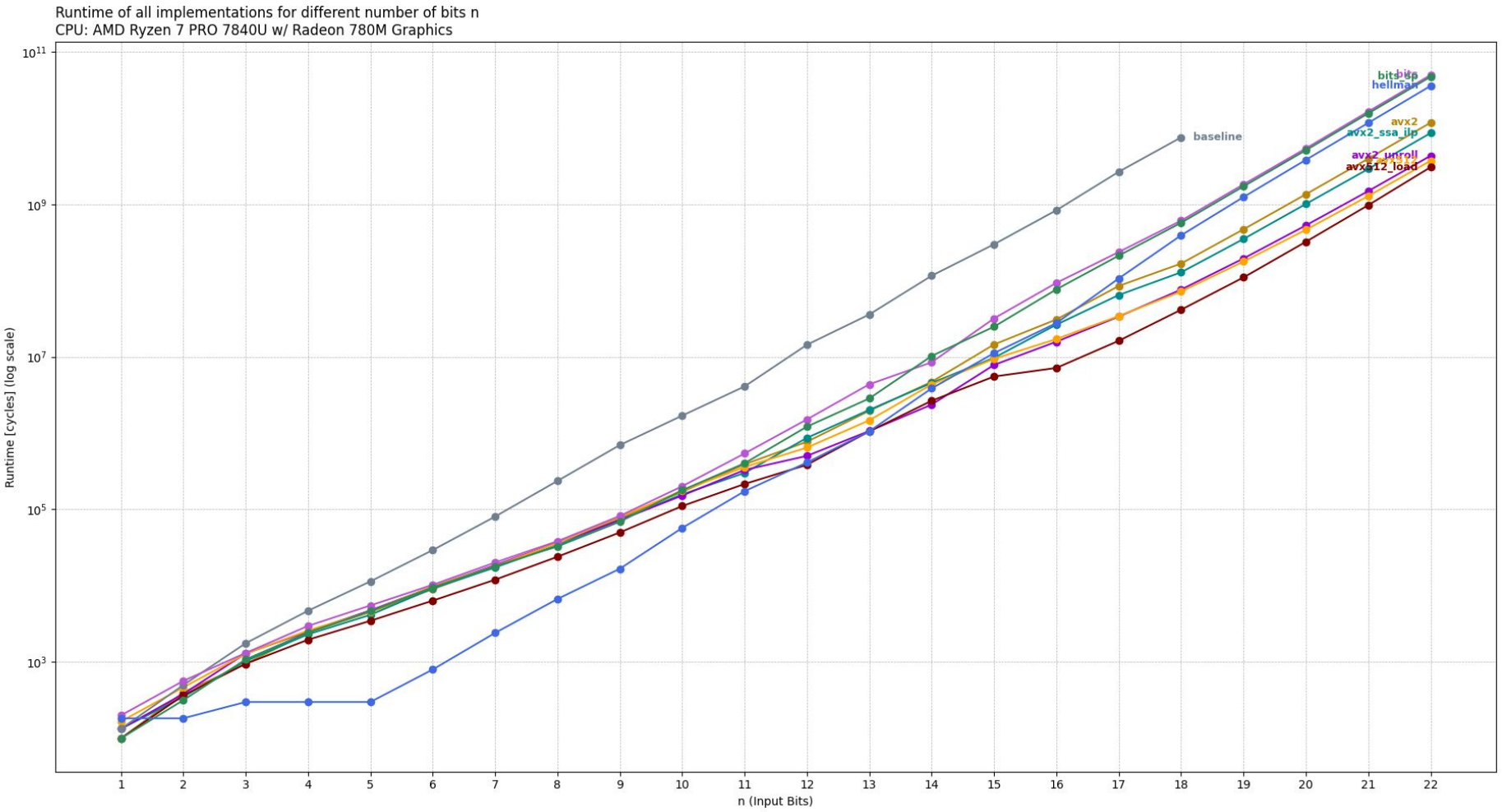
Richard Wohlbold
Aljaz Medic
Aleksa Micanovic
Aleks Stepancic

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Roofline plot



Runtime plot



Appendix: Profiling before loading path

Source Function Stack	CPU Time: Total ▼ Effective Time	CPU Time: Self	Function (Full)	Source File
Total	100.0%	0s		
_start	100.0%	0s	_start	
__libc_start_main_impl	100.0%	0s	__libc_start_main_impl	libc-start.c
main	100.0%	0s	main	main.c
measure_implementations	99.3%	0s	measure_implementations	test.c
prime_implicants_avx2_sp_aljaz	98.6%	0.048s	prime_implicants_avx2_sp...	sp.h
merge_avx2_sp_aljaz	80.8%	0.888s	merge_avx2_sp_aljaz	avx2_sp_alj...
leading_stars	16.1%	0.304s	leading_stars	common.h
[Unknown stack frame(s)]	0.8%	0s	[Unknown stack frame(s)]	
[Unknown stack frame(s)]	0.7%	0s	[Unknown stack frame(s)]	