amadeapewe Sieve: Memory-Compressed Wheel Factorization for ARM64 Architectures

Pavlo Kravchuk (amadeapewe)

amadeapewe@gmail.com

January 22, 2026

Abstract

This paper presents the *amadeapewe* algorithm, a high-performance variant of the Sieve of Eratosthenes. By utilizing a memory-compressed 8-pipe wheel factorization with period $H = 30$, the algorithm achieves significant speedups (up to 3.1×) on modern ARM64 processors. The method optimizes memory layout to improve CPU cache locality and reduces the memory footprint by a factor of 3.75 compared to conventional bitset-based sieves.

1 Introduction

The classical Sieve of Eratosthenes has an algorithmic complexity of $O(N \log \log N)$ and remains one of the most efficient methods for prime number generation. In modern High-Performance Computing (HPC), however, performance is often limited by memory bandwidth and cache latency rather than arithmetic operations.

The *amadeapewe* project focuses on restructuring the memory layout of the sieve to better align with CPU cache lines. Conceptually, the method is equivalent to a wheel-optimized segmented sieve, but it differs in its specific bit-packing strategy and elimination process, leading to improved cache utilization on ARM64 architectures.

2 Architecture: The $H = 30$ Wheel

The core efficiency of the algorithm stems from a wheel with a period of $H = 30$. Only integers n satisfying

$$\gcd(n, 30) = 1$$

are considered candidates for primality. This corresponds to Euler's totient function $\varphi(30) = 8$, yielding the residue set

$$\{1, 7, 11, 13, 17, 19, 23, 29\}.$$

Each of these eight residues is mapped directly to one bit of a single byte. As a result, all multiples of 2, 3, and 5 are eliminated at the data-structure level. This compression allows 30 consecutive integers to be represented using only one byte, significantly increasing cache density and the probability of cache hits during the sieving process.

3 Benchmarking on Apple M1

The algorithm was benchmarked against a standard state-of-the-art segmented wheel sieve implementation on an Apple M1 (ARM64) processor. All code was compiled with aggressive optimization flags (`-O3`).

Scale	**SOTA (ms)**	**amadeapewe (ms)**	**Gain**
Small (10M)	56.06	17.91	3.13×
Medium (100M)	39.92	19.05	2.10×
Large (1B)	41.01	20.46	2.00×

Table 1: Performance comparison on Apple M1 (ARM64).

4 Conclusion

The *amadeapewe* sieve demonstrates that memory-centric optimization of classical algorithms can yield substantial performance improvements on modern hardware. By combining wheel factorization with aggressive memory compression, the method achieves consistent speedups while preserving 100% correctness.

This implementation is suitable for large-scale prime generation and preprocessing tasks in cryptographic and HPC workloads, particularly on ARM64-based systems.

Contacts

Telegram / Instagram: `@amadeapewe`
LinkedIn: linkedin.com/in/amadeapewe