

High Performance Storage of Prime Numbers

J. Schwindling
Independent Researcher

September 2026

Abstract

Although modern computers and algorithms can generate very large prime sets in a relatively short time, storing and distributing the resulting data remains a significant practical problem.

This paper investigates a storage format that exploits the specific properties of prime numbers. Instead of representing every integer up to a limit, the proposed format allocates one bit only to integers that are coprime to a small primorial. The resulting bitmap can subsequently be further compressed using a general purpose compressor such as gzip.

Experiments on a representative interval near 10^{13} show that the compressed representation using 23# (primorial of 23) requires approximately 0.46 bytes per prime. At this rate, storing the first 10^{12} prime numbers would require less than 450 GiB of storage.

1 Introduction

Modern computers can generate prime numbers over very large intervals using highly optimized segmented sieves. Published benchmarks report runtimes ranging from seconds to minutes for numbers up to 10^9 and 10^{12} , depending strongly on the implementation and hardware configuration [1, 2, 3].

The ability to generate a prime set quickly does not eliminate the usefulness of storing the generated data. A stored prime set can be reused without repeating the computation and can be distributed to other computers or used as the backend of an online service.

Several large collections of prime numbers have been published. Don Kostuch’s *Primes to One Trillion*[4], for example, provides all primes below 10^{12} as text files. The uncompressed collection occupies about 486 GB¹, while ZIP compression reduces the total to about 61 GB. For a total of 37 607 912 018 prime numbers below 10^{12} [5], this corresponds to 1.74 bytes to store one prime number.

In 2026, the PrimeForge project reported generating all 37 607 912 018 primes below 10^{12} in less than twelve minutes on a 12-core desktop CPU. Its published binary output is approximately 35 GB and uses gap encoding [3], corresponding to about 0.96 byte per prime number.

An earlier project by Orthallelous generated the first trillion prime numbers, ending at the trillionth prime 29 996 224 275 833. The author reports 13.380 TB of uncompressed data and 642.177 GB after difference encoding and compression [6], improving the average storage per prime number to 0.69 bytes. To our knowledge, the corresponding dataset is not available online.

The present work investigates another prime number representation further reducing the storage needed for large datasets.

¹Storage sizes are reported in the units used by the referenced projects, while storage requirements calculated in this work are expressed in GiB.

2 Existing Prime Data Representations

2.1 Text representation

Decimal text is convenient for human use but inefficient for storage. Furthermore, converting between decimal strings and binary integers adds processing overhead when the data are used by software.

Nevertheless, text representation remains useful for archival and distribution of small datasets because it is universally readable. The Kostuch dataset illustrates the scale of this approach: the complete set below 10^{12} occupies approximately 486 GB as uncompressed text [4], an average of 13.9 bytes per prime number.

2.2 Fixed-width integer representation

The simplest representation stores every prime as an unsigned 64-bit integer (thus 8 bytes per prime number), which is sufficient to represent prime numbers below 10^{19} .

2.3 Gap encoding

Instead of storing

$$p_1, p_2, p_3, \dots,$$

one can store

$$p_1, \quad p_2 - p_1, \quad p_3 - p_2, \dots$$

Although large gaps between prime numbers do happen, more than 99% of the gaps have a value smaller than 128 for prime numbers lower than one billion. The prime gaps can thus be efficiently encoded using variable-length integers.

This approach was used by Orthallous for the first trillion primes [6]. PrimeForge also uses gap encoding in its published dataset [3].

2.4 Bitmaps

A different representation associates one bit with every integer n and sets the bit when n is prime. The file size is therefore

$$\frac{N}{8}$$

bytes for primes below N . For $N = 10^{12}$, the prime fraction is approximately 3.76%, corresponding to about 3.32 bytes per prime number. This representation allocates bits to all integers, including the vast majority of composite numbers. It can thus be improved by a factor 2 by removing a priori all even numbers (assuming that one adds 2 as a prime number at the end).

In [4], an algorithm called *primec* is proposed. Noticing that all primes above 5 end with digits 1, 3, 7 or 9, the author argues that in each range of 20 numbers there are at most 8 primes whose primality can be indicated using 8 bits. Writing any prime p as $p = 20q + r$, the remainder can thus be stored using only one byte. Although the author mentions a test on very large prime numbers, he does not apply this compression to his full dataset.

3 Primorial coprime reduction

Extending the idea of *primec*, let

$$b = p_k\# = \prod_{i=1}^k p_i$$

be the product of the first k primes. Every integer n can be written as

$$n = b \times q + r, \quad 0 \leq r < b.$$

If n is prime and n is larger than the primes included in b , then the residue r must be coprime to b . Consequently, only the $\phi(b)$ residues that are coprime to b need to be represented, where φ is Euler's totient function.

We call this data reduction retaining only integers that are coprime to b *primorial coprime reduction* with modulus b , or *b-coprime reduction*. The resulting sequence is stored as a *b-coprime bitmap*.

The following table shows the reduction factor that can be obtained for various values of b .

Table 1: b -coprime reduction as a function of b .

b	Prime factors	$\varphi(b)$	Fraction	Reduction factor
2	2	1	50.00%	2.000
6	2, 3	2	33.33%	3.000
30	2, 3, 5	8	26.67%	3.750
210	2, 3, 5, 7	48	22.86%	4.375
2 310	2, 3, 5, 7, 11	480	20.78%	4.813
30 030	2, 3, 5, 7, 11, 13	5 760	19.18%	5.214
510 510	2, 3, 5, 7, 11, 13, 17	92 160	18.05%	5.539
9 699 690	2, 3, 5, 7, 11, 13, 17, 19	1 658 880	17.10%	5.847
223 092 870	2, 3, 5, 7, 11, 13, 17, 19, 23	36 495 360	16.36%	6.113

$b = 2$ corresponds to removing all even residues, leading to a reduction of a factor 2 in the space needed to store prime numbers in a bitmap. Removing also the multiples of 3, 5 and 7 leads roughly to another factor 2. Removing also the multiples of 11, 13, 17, 19 and 23 gains an additional factor 1.5. Larger primorials would further reduce the fraction of represented integers, but were not investigated in this study.

This approach is traditionally related to wheel sieve [7, 8], but we use the term coprime reduction to emphasize its role as a storage representation rather than as a sieving optimization.

4 Storage algorithm

We tested the following algorithm to store prime numbers, for a chosen value of b .

4.1 Calculation of the b -coprime bitmap

For a chosen value of b , the integers are processed in consecutive blocks of b integers. Each integer is written as

$$n = bq + r, \quad 0 \leq r < b.$$

For each integer:

- If r is not among the $\phi(b)$ residue classes coprime to b , the integer cannot be prime (apart from the small prime factors of b themselves) and no storage position is allocated to it.
- If r belongs to the allowed residue classes, the integer is tested for primality.
- A bit value of 1 is stored if the integer is prime, and 0 otherwise.

Each block of b consecutive integers therefore produces exactly $\phi(b)$ bitmap positions. Since one bit is used for each position, the resulting raw bitmap requires $\phi(b)/8$ bytes per block of b integers.

For $b = 23\# = 223\,092\,870$, for example, a block of 223 092 870 integers produces 36 495 360 bitmap positions, corresponding to only 4 561 920 bytes of raw storage of the prime numbers contained in this block.

The b -coprime bitmap can be compressed further by using the general-purpose gzip [9] compression tool.

5 Experimental Results

All tests use the same half-open range $[9, 999, 914, 804, 880, 10, 000, 137, 897, 750)$, with a width of 223, 092, 870 integers. This range is around 10^{13} and contains 7 452 769 primes.

Table 2: Compression results for the test interval near 10^{13} .

Format	Raw bytes	Gzip bytes	Gzip/raw	Gzip bytes/prime
PrimeC	11 154 644	5 320 864	0.477	0.714
Storing all bits	27 886 609	6 545 822	0.235	0.878
2-coprime	13 943 305	5 852 171	0.420	0.785
3#-coprime	9 295 537	5 031 536	0.541	0.675
5#-coprime	7 436 429	4 545 258	0.611	0.610
7#-coprime	6 374 082	4 200 013	0.659	0.564
11#-coprime	5 794 620	3 990 081	0.689	0.535
13#-coprime	5 348 880	3 811 432	0.713	0.511
17#-coprime	5 034 240	3 677 780	0.731	0.493
19#-coprime	4 769 280	3 562 771	0.747	0.478
23#-coprime	4 561 920	3 467 426	0.760	0.465

The results are shown in Table 2, including the results of the *primec* algorithm and of storing all bits.

Increasing the coprime modulus from the unreduced bitmap to 23# substantially reduces the storage cost, from 0.878 to 0.465 bytes per prime after gzip compression.

6 Application to a Distributed Prime Database

The storage method described in this work is being implemented in TeraPrimes.org, an open distributed-computing project whose objective is to generate, independently verify, store and distribute the first 10^{12} prime numbers. The project provides a practical testbed for evaluating the storage format, indexing strategy and range-query performance at larger scales.

7 Conclusion

This paper investigates coprime-reduced bitmaps as a storage format for very large collections of prime numbers.

The preliminary results indicate that applying primordial coprime reduction can significantly reduce the amount of data required to store large collections of prime numbers.

After gzip compression, the tested 23#-coprime representation required approximately 0.465 bytes per prime.

At this storage density, storing the first 10^{12} prime numbers would require less than 450 GiB of compressed data.

References

- [1] D. Jacobsen, *Prime Sieve Benchmarks*, 2018–2019. <https://ntheory.org/sieves/benchmarks.html>
- [2] K. Walisch, *primesieve*, <https://github.com/kimwalisch/primesieve>
- [3] BeltUp LLC, *PrimeForge: Segmented-sieve prime engine*, 2026. <https://github.com/BeltUp-LLC/PrimeForge>
- [4] D. Kostuch, *Primes to One Trillion*, Project Gutenberg eBook No. 58225, 2018. <https://www.gutenberg.org/ebooks/58225>
- [5] The OEIS Foundation Inc., *The Online Encyclopedia of Integer Sequences*, Number of primes $< 10^n$, A006880. <https://oeis.org/A006880>
- [6] Orthallelous, *One trillion primes*, 2015. <https://orthallelous.wordpress.com/2015/12/16/one-trillion-primes/>
- [7] P. Pritchard, *A sublinear additive sieve for finding prime numbers*, Comm. ACM 24 (1) (1981) 18–23. <https://dl.acm.org/doi/10.1145/358527.358540>
- [8] P. Pritchard, *Explaining the wheel sieve*, Acta Informat. 17 (1982) 477–485. <https://link.springer.com/article/10.1007/BF00264164>
- [9] GNU Operating System, *GNU Gzip*. <https://www.gnu.org/software/gzip/>