Block Fermat numbers in modular arithmetic

Benjamin Chen, Eugene Zima

University of Waterloo, Waterloo, Canada

June 23, 2025

Introduction

Modular representation is a popular technique to accelerate the arithmetic of computer algebra systems. However, there are overheads involved when this technique is applied:

1. conversion to modular representation (involves division by a modulus)
2. reduction of the intermediate results during computations (division by a modulus)
3. reconstruction from modular representation (involves multiplication and division by moduli and multiplication by computed modular inverses)

Introduction

Modular representation is a popular technique to accelerate the arithmetic of computer algebra systems. However, there are overheads involved when this technique is applied:

1. conversion to modular representation (involves division by a modulus)
2. reduction of the intermediate results during computations (division by a modulus)
3. reconstruction from modular representation (involves multiplication and division by moduli and multiplication by computed modular inverses)

The choice of special-form moduli can help to reduce the overhead. We demonstrate that popular choice of Mersenne type moduli can be outperformed by selecting the moduli of Fermat type.

Mersenne

Mersenne type moduli are of the form $(2^n - 1)$:
A popular choice dating back more than 60 year with multiple generalizations (Schönhage [4], Knuth [3], and Fraenkel [1])

Mersenne

Mersenne type moduli are of the form $(2^n - 1)$:
A popular choice dating back more than 60 year with multiple generalizations (Schönhage [4], Knuth [3], and Fraenkel [1])
Relative primality of $2^n - 1$ and $2^m - 1$ is guaranteed by selecting relatively prime exponents n and m.

Mersenne

Mersenne type moduli are of the form $(2^n - 1)$:
A popular choice dating back more than 60 year with multiple generalizations (Schönhage [4], Knuth [3], and Fraenkel [1])
Relative primality of $2^n - 1$ and $2^m - 1$ is guaranteed by selecting relatively prime exponents n and m.
Overhead 1 and 2 are improved since division by $2^n - 1$ is linear in bit-length of the input.
Overhead 3 is slightly improved as multiplication by $2^n - 1$ is linear in bit-length of the result.
Computing inverses and multiplication by inverses remain a part of overhead 3...

Fermat

Fermat type moduli are of the form $(2^n + 1)$:
Mostly "ignored" until Zima and Steward [5] found that shifted scheme of Fermat moduli gives simple closed-form inverses with three terms. However, moduli with this property are very imbalanced in bit-length.

Fermat

Fermat type moduli are of the form $(2^n + 1)$:
Mostly "ignored" until Zima and Steward [5] found that shifted scheme of Fermat moduli gives simple closed-form inverses with three terms. However, moduli with this property are very imbalanced in bit-length.
Relative primality of $2^n + 1$ and $2^m + 1$ is guaranteed by selecting exponents n and m with different binary valuation.

Fermat

Fermat type moduli are of the form $(2^n + 1)$:
Mostly "ignored" until Zima and Steward [5] found that shifted scheme of Fermat moduli gives simple closed-form inverses with three terms. However, moduli with this property are very imbalanced in bit-length.
Relative primality of $2^n + 1$ and $2^m + 1$ is guaranteed by selecting exponents n and m with different binary valuation.
Overhead 1 and 2 are improved since division by $2^n + 1$ is linear in bit-length of the input.
Overhead 3 is significantly improved as multiplication by $2^n + 1$ is linear in bit-length of the result. Also inverses have sparse pattern and multiplication by inverses is linear in bit-length of the result.

Mersenne vs Fermat (motivational example)

Consider a simple moduli set of size 2, $\{m_1, m_2\}$: Given $u_1 = u \bmod m_1, u_2 = u \bmod m_2$, by CRT and Garner's algorithm:

$$u = u_1 + ((u_2 - u_1)M \bmod m_2)\, m_1,$$

with $0 \leq u < m_1 m_2$ where $M = m_1^{-1} \bmod m_2$.

Mersenne vs Fermat (motivational example)

Consider a simple moduli set of size 2, $\{m_1, m_2\}$: Given $u_1 = u \bmod m_1, u_2 = u \bmod m_2$, by CRT and Garner's algorithm:

$$u = u_1 + ((u_2 - u_1)M \bmod m_2)\, m_1,$$

with $0 \leq u < m_1 m_2$ where $M = m_1^{-1} \bmod m_2$.
Reconstruction involves

1. (pre-)computing inverse M
2. multiplications by M and by m_1
3. division by m_2
4. addition and subtraction (linear time in bit-length of the result)

Mersenne vs Fermat

Consider two Mersenne type moduli: $m_1 = 2^{23} - 1$, $m_2 = 2^{17} - 1$ with $M = 2^{12} + 2^6 + 1$.
Also consider two Fermat type moduli: $m_1 = 2^{24} + 1$, $m_2 = 2^{16} + 1$ with $M = 2^{15} + 2^7 + 1 = 2^{16-1} + 2^{8-1} + 1$.

Mersenne vs Fermat

Consider two Mersenne type moduli: $m_1 = 2^{23} - 1$, $m_2 = 2^{17} - 1$ with $M = 2^{12} + 2^6 + 1$.
Also consider two Fermat type moduli: $m_1 = 2^{24} + 1$, $m_2 = 2^{16} + 1$ with $M = 2^{15} + 2^7 + 1 = 2^{16-1} + 2^{8-1} + 1$.
They seem to have comparable performance for fixed-range computation as reconstruction is linear in the bit-length of the inverse.

Mersenne vs Fermat

However, if our inputs come in various sizes such that we need to dynamically adjust the range of our moduli set:
Suppose we want to increase the representable range by a factor of 10:

- Mersenne: Recomputation is needed, and the sparsity of inverses is not guaranteed.
  One possible new Mersenne type moduli set:
  $m_1 = 2^{239} - 1$, $m_2 = 2^{161} - 1$ with
  $M = 2^{159} + 2^{156} + 2^{154} + \cdots + 2^3 + 1$ (64 terms, dense!).

Mersenne vs Fermat

However, if our inputs come in various sizes such that we need to dynamically adjust the range of our moduli set:
Suppose we want to increase the representable range by a factor of 10:

- Mersenne: Recomputation is needed, and the sparsity of inverses is not guaranteed.
  One possible new Mersenne type moduli set:
  $m_1 = 2^{239} - 1$, $m_2 = 2^{161} - 1$ with
  $M = 2^{159} + 2^{156} + 2^{154} + \cdots + 2^3 + 1$ (64 terms, dense!).
- Fermat: Existing moduli set can be scaled up:
  $m_1 = 2^{240} + 1$, $m_2 = 2^{160} + 1$ with
  $M = 2^{159} + 2^{79} + 1 = 2^{160-1} + 2^{80-1} + 1$ (sparsity is preserved)

Fermat polynomials and their properties

Was the previous example a fluke? It was not.
In fact any two relatively prime Fermat type moduli are scalable (similarly to the example shown).

Fermat polynomials and their properties

Was the previous example a fluke? It was not.
In fact any two relatively prime Fermat type moduli are scalable (similarly to the example shown).
Some facts about Fermat type moduli:

- Scaling (i.e. when 2 is replaced by 2^c for a natural c) preserves the relative primality of two moduli
- Scaling preserves the sparsity of the inverses
- For any $b \in \mathbb{N}$, there exist a set of b pairwise relatively prime and scalable moduli of Fermat type with balanced bit-size

Fermat polynomials and their properties

Was the previous example a fluke? It was not.
In fact any two relatively prime Fermat type moduli are scalable (similarly to the example shown).
Some facts about Fermat type moduli:

- Scaling (i.e. when 2 is replaced by 2^c for a natural c) preserves the relative primality of two moduli
- Scaling preserves the sparsity of the inverses
- For any $b \in \mathbb{N}$, there exist a set of b pairwise relatively prime and scalable moduli of Fermat type with balanced bit-size

In order to prove these facts, it is more convenient to study Fermat polynomials first.
Observation: Given a Fermat type polynomial $f(x) = x^n + 1$, it links naturally to its corresponding Fermat type modulus $f(2) = 2^n + 1$ and scaled Fermat type modulus $f(2^c) = 2^{cn} + 1$.

Fermat polynomials and their properties

Consider two Fermat type polynomials $f(x) = x^n + 1$ and $g(x) = x^m + 1$ in $\mathbb{Q}[x]$.

Existence of Fermat Moduli Sets:

- $$\gcd(x^n + 1, x^m + 1) = \begin{cases} 1, & \nu_2(n) \neq \nu_2(m) \\ x^{\gcd(n,m)} + 1, & \text{otherwise} \end{cases}$$

  $\nu_2(n)$: binary valuation of a natural number n

- If $\gcd(f(x), g(x)) = 1$, then for any integer $\ell \geq 1$ numbers $f(2^\ell)$ and $g(2^\ell)$ are relatively prime.

Fermat polynomials and their properties

Bézout cofactors:

- If $\gcd(f(x), g(x)) = 1$, then non-zero coefficients of Bézout cofactors $s(x)$ and $t(x)$ in equation

$$s(x)f(x) + t(x)g(x) = 1$$

are dyadic with the numerators ± 1 and denominators equal to 2.
Thus, $f(2^\ell) = 2^{\ell n} + 1$ and $g(2^\ell) = 2^{\ell m} + 1$ is a scalable pair of moduli with scalable inverses for any $\ell \geq 2$.

Fermat polynomials and their properties

Bézout cofactors:

- If the cofactors $s(x), t(x)$ have more than 1 term, the difference in adjacent degrees of the terms in the cofactors ordered by the degree is $h = \gcd(n, m)$. Specifically,

$$s(x) = \pm\frac{1}{2}x^{m-h} \pm \frac{1}{2}x^{m-2h} \pm \cdots \pm \frac{1}{2},$$

$$t(x) = \pm\frac{1}{2}x^{n-h} \pm \frac{1}{2}x^{n-2h} \pm \cdots \pm \frac{1}{2}.$$

Fermat polynomials and their properties

Existence of scalable inverses:

- If $\gcd(f(x), g(x)) = 1$ then there exist polynomials $p(x)$ with $\deg p(x) \leq m$ and $q(x)$ with $\deg q(x) \leq n$ having coefficients from the set $\{0, \pm 1/2, \pm 1\}$ such that

$$f(2^\ell)^{-1} \bmod g(2^\ell) = p(2^\ell),$$

$$g(2^\ell)^{-1} \bmod f(2^\ell) = q(2^\ell).$$

The number of set bits in the scaled inverses is the same as the support of polynomials $p(x)$ and $q(x)$ and will be preserved under scaling.

Fermat polynomials and their properties

Existence of scalable inverses:

- If $\gcd(f(x), g(x)) = 1$ then there exist polynomials $p(x)$ with $\deg p(x) \leq m$ and $q(x)$ with $\deg q(x) \leq n$ having coefficients from the set $\{0, \pm 1/2, \pm 1\}$ such that

$$f(2^\ell)^{-1} \bmod g(2^\ell) = p(2^\ell),$$

$$g(2^\ell)^{-1} \bmod f(2^\ell) = q(2^\ell).$$

The number of set bits in the scaled inverses is the same as the support of polynomials $p(x)$ and $q(x)$ and will be preserved under scaling.
For example, if $f(x) = x^{15} + 1$ and $g(x) = x^{20} + 1$ then $q(x) = \frac{1}{2}x^{15} + \frac{1}{2}x^{10} + \frac{1}{2}x^5 + 1$ and for any integer $\ell \geq 1$

$$g(2^\ell)^{-1} \mod f(2^\ell) = q(2^\ell) = 2^{15\ell-1} + 2^{10\ell-1} + 2^{5\ell-1} + 1.$$

Selecting Fermat type moduli

Two Fermat moduli are co-prime iff their exponents have different binary valuations. There are two greedy schemes to generate the exponents of a moduli set of size b:

1. $e_k = 2^b - 2^{k-1}$ for $k = 1, 2, \ldots, b$.
   Example: When $b = 4$,

$$[1111_{(2)}, 1110_{(2)}, 1100_{(2)}, 1000_{(2)}]$$

2. $e_k = 2^{b-1} + 2^{b-k-1}$ for $k = 1, 2, \ldots, b-1$ and $e_b = 2^{b-1}$.
   Example: When $b = 4$,

$$[1100_{(2)}, 1010_{(2)}, 1001_{(2)}, 1000_{(2)}]$$

Selecting better blocks of Fermat type moduli

Also a valid set:

$$[1100_{(2)}, 1010_{(2)}, 1011_{(2)}, 1000_{(2)}]$$

Greedy schemes are not optimal.

Selecting better blocks of Fermat type moduli

Number of terms in inverses:

Given $f(x) = x^n + 1, g(x) = x^m + 1$ with $\gcd(f(x), g(x)) = 1$, the numbers of terms in $s(x), t(x)$, where $s(x)f(x) + t(x)g(x) = 1$, are $m/\gcd(m, n)$ and $n/\gcd(m, n)$.

The numbers of terms in $p(x), q(x)$ (polynomials whose evaluations are scaled inverses) are $m/\gcd(m, n) + 1$ and $n/\gcd(m, n) + 1$.

Selecting better blocks of Fermat type moduli

Comparing the total support of Bézout cofactors:
When $b = 6$:

Scheme	Moduli Exponents	Total Supports
Greedy1	$\{63, 62, 60, 56, 48, 32\}$	289
Greedy2	$\{48, 40, 36, 34, 33, 32\}$	233
Best	$\{63, 56, 48, 42, 36, 32\}$	141

It is worth noting that the total support of Bézout cofactors influences the time of reconstruction.

Selecting better blocks of Fermat type moduli

Exhaustive search with back-tracking to find a set of b exponents with the least total support of Bézout cofactors.

Search results show that the total support of optimal choice grows much slower than the total support of greedy schemes.

b	Best	Greedy1	Greedy2
6	141	289	233
7	279	937	576
8	534	1962	1227
9	1026	4740	3290
10	1935	9479	6433
11	3779	27923	15052
12	7273	46184	30771
13	14441	136310	76090
14	28153	254909	149839
15	55718	510173	339918

Table: Selective Results of Total Support of Different Schemes

Implementation and Benchmark

Integer matrices multiplication

Dim	Bitsize	M Mult	M Overh	M Total	F Mult	F Overh	F Total
8	2^{18}	0.124	0.282	0.407	0.114	0.093	0.207
-	2^{19}	0.287	0.693	0.980	0.297	0.188	0.485
-	2^{20}	0.768	1.583	2.351	0.783	0.376	1.167
16	2^{18}	0.931	1.056	1.988	0.887	0.387	1.274
-	2^{19}	2.229	2.742	4.972	2.271	0.786	3.058
-	2^{20}	5.970	6.338	12.308	6.004	1.571	7.613
32	2^{18}	7.376	4.305	11.683	7.165	1.542	8.709
-	2^{19}	14.882	8.980	23.864	16.124	2.781	18.908
-	2^{20}	39.726	20.806	60.536	39.887	5.652	45.675
64	2^{18}	49.410	14.472	63.890	47.457	5.460	52.925
-	2^{19}	119.274	36.020	155.304	120.622	11.566	132.199
-	2^{20}	318.661	83.032	401.706	318.602	22.857	341.985

Table: Dim - matrix dimension, Bitsize - bitsize of elements of matrices, M - Mersenne-type moduli, F - Fermat-type moduli

Implementation and Benchmark

Dim	Bitsize	M Mult	M Overh	M Total	F Mult	F Overh	F Total
8	2^{20}	0.768	1.583	2.351	0.783	0.376	1.167

Table: Dim - matrix dimension, Bitsize - bitsize of elements of matrices, M - Mersenne-type moduli, F - Fermat-type moduli

In this highlighted row, the total bitlength of the inverses is about $2^{20} = 1,048,576$ while only about 150 bits are set in total in all scaled-up inverses.

Implementation and Benchmark

Dim	Bitsize	M Mult	M Overh	M Total	F Mult	F Overh	F Total
32	2^{18}	7.376	4.305	11.683	7.165	1.542	8.709
-	2^{19}	14.882	8.980	23.864	16.124	2.781	18.908
-	2^{20}	39.726	20.806	60.536	39.887	5.652	45.675

Table: Dim - matrix dimension, Bitsize - bitsize of elements of matrices, M - Mersenne-type moduli, F - Fermat-type moduli

The overhead in Fermat numbers grows linearly as the inverses have the same number of bits set. Doubling the size means doubling the work. (Reduction and Reconstruction)

Implementation and Benchmark

Dim	Bitsize	M Mult	M Overh	M Total	F Mult	F Overh	F Total
32	2^{18}	7.376	4.305	11.683	7.165	1.542	8.709
-	2^{19}	14.882	8.980	23.864	16.124	2.781	18.908
-	2^{20}	39.726	20.806	60.536	39.887	5.652	45.675

Table: Dim - matrix dimension, Bitsize - bitsize of elements of matrices, M - Mersenne-type moduli, F - Fermat-type moduli

For Mersenne numbers, the amount of overhead for reduction is doubled, but the reconstruction is not (computing extended Euclidean algorithm naively is $O(n^2)$ where n is the number of bits in the number, also the inverse can be dense with bitlength doubled)

Implementation and Benchmark

Comments:

- Mersenne type moduli and Fermat type moduli have similar performance in terms of multiplication.
- Fermat type moduli outperforms Mersenne type moduli due to the scalability and the sparsity of the inverses.
  - It is also possible to precompute and store the inverses of Mersenne type moduli, but such inverses cannot be dynamically scaled like Fermat type moduli.
  - Even if inverses are pre-computed for Mersenne type moduli – Fermat type moduli reconstruction is still faster.
  - Sparse inverses also lead to huge memory savings.

Conclusion and open question

Block Fermat-type moduli

- Scale up a given moduli set without recomputation
- The moduli being only slightly unbalanced
- Significant reduction in the overhead compared to Mersenne-type moduli
- Drawback: the size of moduli grows exponentially with block size b, which makes only "small" sets of moduli practical
  - Best suited for situations where only *a few* large moduli are needed. For example, a two-layer modular arithmetic implementation. (Chen, Li, and Zima [2])

Conclusion and open question

Finding Fermat-type moduli

- Exhaustive search with pruning
- Greedy schemes of constructing exponents provide sets with reasonably good characteristics

Open question: Is there a better algorithm to find close-to-optimal blocks of balanced-in-size exponents that guarantees pairwise relative primality of the moduli?

Bibliography I

[1] Aviezri S. Fraenkel. "The Use of Index Calculus and Mersenne Primes for the Design of a High-Speed Digital Multiplier". In: *Journal of the ACM* 8.1 (Jan. 1961), pp. 87–96. ISSN: 0004-5411, 1557-735X. DOI: 10.1145/321052.321057. URL: https://dl.acm.org/doi/10.1145/321052.321057 (visited on 09/04/2024).

[2] Benjamin Chen, Yu Li, and Eugene Zima. "On a Two-Layer Modular Arithmetic". In: *ACM Communications in Computer Algebra* 57.3 (Sept. 2023), pp. 133–136. ISSN: 1932-2240. DOI: 10.1145/3637529.3637534. URL: https://dl.acm.org/doi/10.1145/3637529.3637534 (visited on 09/04/2024).

[3] Donald Ervin Knuth. *The art of computer programming. 2: Seminumerical algorithms*. Reading, Mass: Addison-Wesley, 1969. 624 pp. ISBN: 978-0-201-03802-6.

Bibliography II

[4] A. Schönhage. "Multiplikation großer Zahlen". In: *Computing* 1.3 (Sept. 1966), pp. 182–196. ISSN: 0010-485X, 1436-5057. DOI: 10.1007/BF02234362. URL: http://link.springer.com/10.1007/BF02234362 (visited on 05/29/2023).

[5] E. V. Zima and A. M. Stewart. "Cunningham numbers in modular arithmetic". In: *Programming and Computer Software* 33.2 (Mar. 2007), pp. 80–86. ISSN: 0361-7688, 1608-3261. DOI: 10.1134/S0361768807020053. URL: http://link.springer.com/10.1134/S0361768807020053 (visited on 07/21/2024).