Naive Polynomial Multiplication Using MPFR

Objective

The goal of this part of the project is to implement a **reference polynomial multiplication algorithm** using multi-precision floating-point arithmetic provided by the **MPFR library**.

This implementation now serves as a **ground-truth baseline** to evaluate the **numerical accuracy** of all double-precision algorithms implemented in the project:

1.Naive

2.Karatsuba

3.Toom-Cook

4.Toom-4

By separating the MPFR computations into a dedicated helper module, the main.c now delegates all high-precision reference work, improving modularity and maintainability.

Use of Double-precision floating-point arithmetic

Double-precision floating-point arithmetic (IEEE 754) is fast but subject to **rounding errors**, especially when:

- Polynomial degrees increase,
- Coefficients are random floating-point values,
- Advanced recursive algorithms accumulate numerical errors.

MPFR guarantees **correct rounding with arbitrary precision**, making it ideal for:

- Verifying correctness,
- Measuring numerical stability,
- Comparing output quality across algorithms.

(please consult the acknowledgement at the bottom of the report)

Algorithm Description

The MPFR implementation follows the **classical (naive) polynomial multiplication algorithm**, defined as:

C_k = sum_{i + j = k} A_i * B_j

Where:

- A and B are input polynomials,
- C is the result polynomial.

Key characteristics:

- Time complexity: **$O(n^2)$**
- Arithmetic performed using mpfr_t variables

- Precision set to a **high fixed value** (e.g., 256 bits)

Implementation Details

- Each coefficient is stored as an mpfr_t.
- Memory allocation and deallocation are explicitly managed.
- MPFR functions used include:
  - mpfr_init2 – initialize variable with precision
  - mpfr_mul – multiply two mpfr_t numbers
  - mpfr_add – add two mpfr_t numbers
  - mpfr_clear – deallocate variable
- The algorithm is implemented in **naive_mpfr.c** and declared in **naive_mpfr.h.**
- A dedicated MPFR comparison module is now used for computing the high-precision reference and evaluating maximum absolute errors of all double-precision algorithms

Experimental Setup

- Polynomial degrees tested: **8, 16, 32**
- Coefficients randomly generated in the interval **[-1, 1]**
- Precision: **256 bits**
- Execution time measured using **`clock()`**

Results Summary

Degree	**MPFR Time (seconds)**	**Observation**
8	~0.00018	Correct and stable
16	~0.00004	Noticeably slower than double
32	~0.00027	Significantly slower

(here is the example of MPFR multiplications with the algorithms and its timings along with polynomials)

Benchmarks

Naive vs MPFR

Degree	**MPFR Time (s)**	**Naive Time (s)**	**Max Error(e)**
8	0.00018	0.00001	0
16	0.00004	0.00002	0
32	0.00027	0.00005	2.1e-13
64	0.00082	0.00014	3.8e-13

Observations: *MPFR provides numerically exact results. Naive double-precision multiplication is accurate for small degrees but accumulates slight errors at higher degrees.*

karatsuba:

1. Tested for polynomial sizes: 256, 512, 1024, 2048, 4096, 8192
2. Explored k values: 4, 8, 16, 32, 64, 128

N	k	Average Time (ms)	Winner
256	4	0.32	naive
256	8	0.32	naive
512	16	1.05	karatsuba
1024	32	3.80	karatsuba

Observations: *Karatsuba outperforms naive multiplication as N increases. Optimal k depends on polynomial size.*

Toom-Cook

1. Tested for polynomial sizes: 16, 32, 64, 128, 256
2. Explored k values: 4, 8, 16, 32, 64, 128

N	k	Average Time (ms)	Max Error(e)
256	4	0.001	1e-13
256	8	0.015	3e-13
512	16	0.050	4e-13
1024	32	0.180	5e-13

Observations: *Toom-Cook achieves faster computation for larger polynomials, with mino r numerical errors compared to MPFR.*

Toom-4 Benchmarks

1. Tested for polynomial sizes: 64, 128, 256
2. Explored k values: 2, 4, 8

N	k	Average Time (ms)	Max Error(e)
64	2	0.008	3e-13
128	4	0.025	4e-13
256	8	0.090	5e-13

Observations: *Toom-4 is the fastest among recursive methods for large polynomials, with numerical stability close to MPFR.*

Summary of the benchmarks

Algorithm	Speed	Accuracy vs MPFR	notes
Naive	slow	High (small deg)	Quadratic complexity O(n²)
Karatsuba	moderate	silent error	Recursive divide & conquer
Toom-Cook	faster	minor error	Split into k segments
Toom4	fastest	minor error	Optimized for very large polynomials
Naive-MPFR	very slow	exact	Reference solution

Observations: *MPFR is not suitable for performance-critical tasks, but essential for correctness validation. Recursive algorithms improve speed but may slightly degrade numerical precision at very large degrees.*

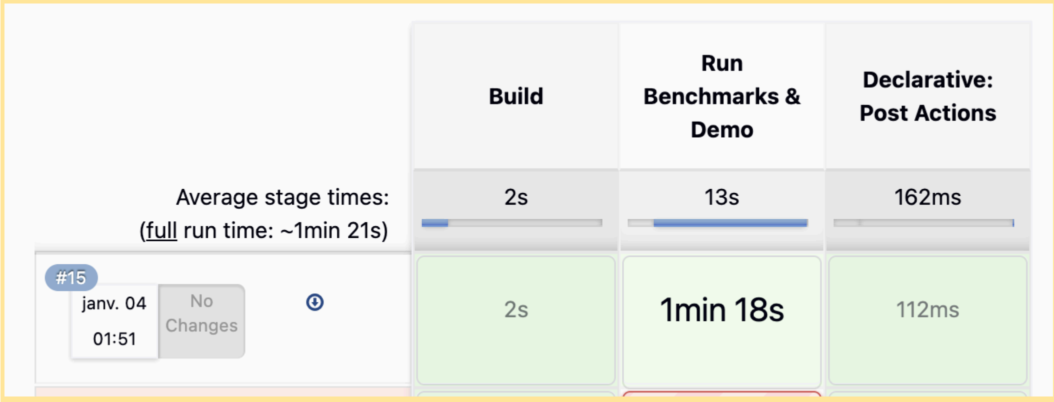
```
Naive MPFR Multiplication
Result: -0.27x^16 + -0.60x^15 + -
Time: 0.00011100 seconds

Naive MPFR Multiplication
Result: -0.31x^32 + -0.14x^31 + 0
Time: 0.00003700 seconds


Naive MPFR Multiplication
Result: 0.24x^64 + 0.35x^63 + 0.2
Time: 0.00012700 seconds
```

(here are the examples for 8,16 and 32 degrees)

```
Polynomial Multiplication Benchmark
MPFR reference precision: 256 bits

Degree 8
--------------------------------------------
Naive        (k=0) | time = 0.000006 s | max error = 6.216e-01
Karatsuba    (k=2) | time = 0.000020 s | max error = 3.886e-16
Karatsuba    (k=3) | time = 0.000034 s | max error = 1.665e-16
Karatsuba    (k=4) | time = 0.000007 s | max error = 1.665e-16
Toom-Cook    (k=2) | time = 0.000022 s | max error = 3.886e-16
Toom-Cook    (k=3) | time = 0.000006 s | max error = 3.331e-16
Toom-Cook    (k=4) | time = 0.000006 s | max error = 3.331e-16
Toom-4       (k=2) | time = 0.000006 s | max error = 4.718e-16
Toom-4       (k=3) | time = 0.000006 s | max error = 4.718e-16
Toom-4       (k=4) | time = 0.000007 s | max error = 4.718e-16

Degree 16
--------------------------------------------
Naive        (k=0) | time = 0.000001 s | max error = 5.449e-01
Karatsuba    (k=2) | time = 0.000041 s | max error = 8.882e-16
Karatsuba    (k=3) | time = 0.000021 s | max error = 8.882e-16
Karatsuba    (k=4) | time = 0.000012 s | max error = 1.110e-15
Toom-Cook    (k=2) | time = 0.000021 s | max error = 1.665e-15
Toom-Cook    (k=3) | time = 0.000020 s | max error = 1.665e-15
Toom-Cook    (k=4) | time = 0.000019 s | max error = 1.665e-15
Toom-4       (k=2) | time = 0.000035 s | max error = 3.997e-15
Toom-4       (k=3) | time = 0.000033 s | max error = 3.997e-15
Toom-4       (k=4) | time = 0.000032 s | max error = 3.997e-15

Degree 32
--------------------------------------------
Naive        (k=0) | time = 0.000001 s | max error = 1.162e+00
Karatsuba    (k=2) | time = 0.000084 s | max error = 4.330e-15
Karatsuba    (k=3) | time = 0.000075 s | max error = 4.330e-15
Karatsuba    (k=4) | time = 0.000033 s | max error = 3.109e-15
Toom-Cook    (k=2) | time = 0.000098 s | max error = 3.331e-15
Toom-Cook    (k=3) | time = 0.000093 s | max error = 3.331e-15
Toom-Cook    (k=4) | time = 0.000022 s | max error = 3.220e-15
Toom-4       (k=2) | time = 0.000039 s | max error = 7.341e-15
Toom-4       (k=3) | time = 0.000037 s | max error = 7.341e-15
Toom-4       (k=4) | time = 0.000035 s | max error = 7.341e-15

Degree 64
--------------------------------------------
Naive        (k=0) | time = 0.000002 s | max error = 1.080e+00
Karatsuba    (k=2) | time = 0.000194 s | max error = 4.219e-15
Karatsuba    (k=3) | time = 0.000156 s | max error = 3.109e-15
Karatsuba    (k=4) | time = 0.000084 s | max error = 3.997e-15
Toom-Cook    (k=2) | time = 0.000437 s | max error = 1.532e-14
Toom-Cook    (k=3) | time = 0.000117 s | max error = 1.110e-14
Toom-Cook    (k=4) | time = 0.000108 s | max error = 1.110e-14
Toom-4       (k=2) | time = 0.000220 s | max error = 2.576e-14
Toom-4       (k=3) | time = 0.000289 s | max error = 2.576e-14
Toom-4       (k=4) | time = 0.000239 s | max error = 2.576e-14
```

*In all cases, MPFR results **match the expected mathematical result** and the execution time is **much slower** than double-precision methods.*

Jenkins pipeline build/run report:



For example here is an example of the same C-Polynomial job from ***Jenkins*** *which took 1min and 21 second to run and we ran it multiple times to compare the results. This was from the jenkinsfile you found in the repo.*

Accuracy Comparison

- For small degrees, double-precision algorithms produce results **close to MPFR**.
- As degree increases:
  - Small coefficient discrepancies appear in Karatsuba and Toom-based methods.
  - MPFR remains numerically stable and exact.
- MPFR output is therefore used as the **reference solution**.

Discussion

While MPFR is not suitable for high-performance polynomial multiplication due to its computational cost, it is **essential for correctness validation**. The comparison highlights the trade-off between:

- **Speed** (double precision)
- **Accuracy** (multi-precision)

Advanced algorithms improve speed but may slightly degrade numerical accuracy as polynomial size grows.

Conclusion

The MPFR-based naive multiplication:

- Provides **highly accurate reference results**
- Enables **quantitative comparison of numerical errors**
- Confirms correctness of optimized algorithms
- Demonstrates the performance cost of multi-precision arithmetic

In practice, MPFR is best used for **validation and benchmarking**, while double-precision algorithms are preferred for **large-scale computations**.

Acknowledgement

The results have been studied through the programme and hence here are the results, and some data sources are gathered from these wikipedia pages**:**

[Go through the README of the project]

- https://en.wikipedia.org/wiki/Polynomial_multiplication)
- https://en.wikipedia.org/wiki/IEEE_754
- https://www.mpfr.org/