Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Национальный исследовательский университет «МЭИ»

|  |  |  |  |
| --- | --- | --- | --- |
| Институт: | ИВТИ | Кафедра: | МКМ |
| Направление подготовки/специальность: | | 01.04.02 Прикладная математика и информатика | |

**ЗАДАНИЕ НА ПРАКТИКУ**

|  |  |
| --- | --- |
| **Наименование практики:** | Учебная практика: технологическая (проектно-технологическая) практика |
|  |  |
| **Студент:** | Романов Андрей Эдуардович |
|  | *(Фамилия, имя, отчество (при наличии) полностью)* |
| **Группа:** | А-14м-24 |
|  | *(номер учебной группы)* |
| **Место прохождения практики:** | каф. Математического и компьютерного моделирования |
|  | *(наименование предприятия, организации, учреждения, подразделения МЭИ в соответствии с приказом о направлении на практику)* |
| **Сроки практики:** | 02.09.2024-23.12.2024; |
|  | *(в соответствии с приказом о направлении на практику)* |

|  |
| --- |
| **Содержание задания:** |
| 1.Вводный инструктаж на месте прохождения практики |
| 2.Поиск источников информации по теме исследования |
| 3.Систематизировать и проанализировать найденную информацию по теме исследования |
| 4.Обосновать актуальность темы исследования |
| 5.Поставить цель и сформулировать задачи исследования |
| 6.Выполнить иные задания руководителя практики |
| 7.По результатам практики составить индивидуальный письменный отчет по практике |
| *(вопросы, подлежащие изучению в соответствии с планируемыми результатами обучения,*  *заполняются руководителем практики от МЭИ)* |

**По результатам прохождения практики студент оформляет отчет по установленной форме.**

|  |  |  |
| --- | --- | --- |
| Руководитель практики  (от МЭИ) | 15.09.2024 | А.М. Бирюков |
|  | *(дата)* | (*Фамилия и инициалы*) |

|  |  |  |
| --- | --- | --- |
| Студент | 23.09.2024 | А.Э. Романов |
|  | *(дата)* | (*Фамилия и инициалы*) |
| СОГЛАСОВАНО:  Руководитель практики от | каф. Математического и компьютерного моделирования | |
|  | *(наименование предприятия, организации, учреждения, подразделения МЭИ в соответствии с приказом о направлении на практику)* | |
| доцент | 15.09.2024 | А.М. Бирюков |
| *(должность)* | *(дата)* | (*Фамилия и инициалы*) |

В рамках практики была составлена задача: рассмотреть и произвести сравнение библиотек для работы с большими числами в языке программирования С++.

Большие числа используются в различных научных областях, технологиях, где исследуемые, обрабатываемые числа выходят за пределы стандартных типов данных, например , обладающий размером 64 бит, или когда необходима высокая точность, которую не может дать с длинной мантиссы 52 бит.

1. **Обзор библиотек**
   1. **GMP (MPIR)**

GMP (GNU Multiple Precision Arithmetic Library) – библиотека для работы с большими числами и числами произвольной точности. Используется для выполнения высокоскоростных операций с целыми числами, с числами с плавающей запятой и рациональными числами.

Библиотека написана на языке программирования C, что позволяет ей оставаться одной из самых высокопроизводительных среди аналогов.

Библиотека является кроссплатформенной и поддерживает работу на различных операционных системах, включая Linux, macOS, Windows и на различных архитектурах процессоров.

Работа с GMP заключается в использовании представленных типов данных и функций. Представленные типы данных: *mpz\_t*, *mpf\_t*, *mpq\_t* для целых, вещественных и рациональных чисел соответственно. Функции, реализующие операции, производимые над числами, содержат в себе префикс, обозначающий тип данных операндов, над которыми производятся операция. Например, для сложения двух целых чисел необходимо вызвать функцию:

Функция принимает в себя целые операнды *op1*, *op2* и сохраняет результат в *rop*. Операнд *op2* может обладать типом *unsigned long int*, что позволяет складывать числа с константами.

Помимо простых операций, таких как сложение, умножение, деление и так далее, в библиотеке есть функции, реализующие алгебраические операции, например НОД, вычисление корней, факторизация.

Недостатком библиотеки следует выделить сложность установки на компьютеры под управлением Windows, что нивелируется использованием популярной библиотеки MPIR, основанной на GMP.

В данной работе используется именно MPIR, так как тестирование и анализ выполнялись на ОС Windows.

* 1. **NTL**

NTL (Number Theory Library) – библиотека для работы с большими числами, используемая в теории чисел, криптографии и других научных областях, где требуется работа с большими числами и полиномами.

Библиотека написана на языке программирования C++ и предоставляет удобный интерфейс для работы с ней.

Работа с NTL осуществляется с использованием представленных классов, функций и перегрузок базовых операций языка. Представленные типы данных: *ZZ, QQ, RR, ZZX* для целых, рациональных, вещественных чисел и целочисленных полиномов соответственно. Описанные выше типы данных являются классами, соответственно, относительно объектов можно вызывать некоторые методы, например *abs(), negate()* для нахождения модуля и отрицания *SetPrecision(long a)* – установка точности для вещественных чисел. Так же, стоит отметить, что реализация типов данных в виде классов избавляет разработчика от необходимости прослеживать жизненный цикл создаваемых им объектов. Библиотека сама инициализирует, удаляет память посредством конструкторов и деструкторов. Простые операции, такие как сложение, умножение, деление и так далее реализованы с помощью перегрузок операций, что делает написание кода при помощи этой библиотеки простым и крайне удобным.

Помимо работы с числами библиотека предоставляет обширный инструментарий для работы с полиномами, матрицами, что делает возможным использование библиотеки для решения задач линейной алгебры: решения СЛАУ, нахождения собственных значений матриц.

Недостатком NTL можно выделить низкую производительность, по сравнению с GMP. Вычисления производимые при помощи NTL будут в разы медленнее, чем те, что производятся при помощи GMP.

1. **Тестирование библиотек**

В рамках задания исследовательской работы было необходимо произвести тестирование производительности GMP и NTL. Для оценки производительности был написан небольшой алгоритм, проводящий измерения времени выполнения операций на массиве из 1000, 10000 элементов. Некоторые операции, например , не представлены в каждой из библиотек, следовательно, выбирались именно те операции, которые реализованы как в GMP, так и в NTL.

Исследуемые операции:

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1. Список исследуемых операций | | | |
| Операция | Тип данных | Функция GMP | Функция NTL |
| Сложение | Целое | mpz\_add | operator+ |
| Вещественный | mpf\_add |
| Умножение | Целое | mpz\_mul | operator\* |
| Вещественный | mpf\_mul |
| Деление | Целое | mpz\_div | operator/ |
| Вещественный | mpf\_div |
| Возведение в степень | Целое | mpz\_pow\_ui | power |
| Вещественный | mpf\_pow\_ui |
| Квадратный корень | Целое | mpz\_root\_ui | SqrRoot |
| Вещественный | mpf\_sqrt |
|  |  |  |  |

В качестве второго операнда для бинарных операций использовались элементы того же массива, следовательно, вычисление таких операций заняло в N раз больше времени, в то время как унарные операции применялись относительно каждого элемента единожды.

Степень возведения, было принято решение, выбрать 2 ввиду того, что взятие n-го корня из числа не поддерживалась всеми типами данных исследуемых библиотек, в то время как вычисление квадратного корня было допустимо.

В ходе написания алгоритма тестирования было обнаружено, что библиотеки не могут работать совместно, следовательно, при работе с ними необходимо выбрать одну. В нашем случае были созданы два исполняемых файла.

Алгоритм поддерживает параметры запуска. Таким образом, запуская программу из консоли можно указать некоторые параметры запуска, такие как: путь сохраняемого файла с результатами, размер массива операндов, n-степень возведения, взятия корня.

1. **Анализ и сравнение результатов тестирования**

Алгоритм тестирования, по ходу выполнения работы, создает текстовые файлы, содержащие в себе данные об исследуемых операциях. Отображаемые данные:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 2. Структура файла результатов | | | | | | |
| Тип операнда | int | | | | | float |
| Разрядность | 128 | 256 | 512 | | 1024 |  |
| Размер массива | 1000 | | | 10000 | | |
| Операция | (Таблица 1) | | | | | |
| Время выполнения | секунды | | | | | |
|  |  | | | | | |

Для интерпретации результатов в табличной, графической, форме был написан скрипт на языке Python при помощи библиотек *re* (регулярные выражения), *pandas* (табличная форма) и *matplotlib* (графическое представление).

Рассмотрим результаты выполнения программ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 3. Результаты выполнения GMP на 1000 элементов массива | | | | | |
|  | float | int\_128 | int\_256 | int\_512 | int\_1024 |
| add | 0.078 | 0.02 | 0.029 | 0.048 | 0.083 |
| div | 0.41 | 0.193 | 0.247 | 0.322 | 0.411 |
| mul | 0.188 | 0.079 | 0.186 | 0.533 | 1.783 |
| powui | 0.001 | 0.001 | 0.001 | 0.001 | 0.002 |
| sqrt | 0.0 | 0.001 | 0.003 | 0.004 | 0.005 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 4. Результаты выполнения NTL на 1000 элементов массива | | | | | |
|  | float | int\_128 | int\_256 | int\_512 | int\_1024 |
| add | 0.875 | 0.333 | 0.348 | 0.41 | 0.483 |
| div | 2.766 | 0.59 | 0.704 | 0.941 | 1.45 |
| mul | 1.28 | 0.945 | 2.208 | 6.24 | 19.227 |
| powui | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| sqrt | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 5. Результаты выполнения GMP на 10000 элементов массива | | | | | |
|  | float | int\_128 | int\_256 | int\_512 | int\_1024 |
| add | 7.918 | 2.059 | 2.991 | 4.844 | 8.163 |
| div | 41.564 | 19.421 | 24.932 | 32.514 | 40.417 |
| mul | 19.012 | 7.991 | 17.863 | 51.951 | 177.407 |
| powui | 0.005 | 0.002 | 0.003 | 0.007 | 0.015 |
| sqrt | 0.005 | 0.012 | 0.023 | 0.038 | 0.05 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 6. Результаты выполнения NTL на 10000 элементов массива | | | | | |
|  | float | int\_128 | int\_256 | int\_512 | int\_1024 |
| add | 88.523 | 34.252 | 36.055 | 41.954 | 49.865 |
| div | 283.145 | 60.876 | 72.522 | 96.84 | 148.47 |
| mul | 131.049 | 96.713 | 226.134 | 643.532 | 1940.221 |
| powui | 0.1 | 0.099 | 0.099 | 0.1 | 0.099 |
| sqrt | 0.099 | 0.1 | 0.1 | 0.1 | 0.1 |
|  |  |  |  |  |  |

Из результатов, представленных в таблицах, уже можно заметить, что программа написанная с GMP выполняется в разы быстрее программы с NTL. Также, можно заметить, что унарные операции возведения в квадрат, взятие квадратного корня для библиотеки NTL выполняются одинаково медленно для любых типов данных и медленнее чем для GMP.

Рассмотрим графики исследуемых результатов на 10000 элементов массива:

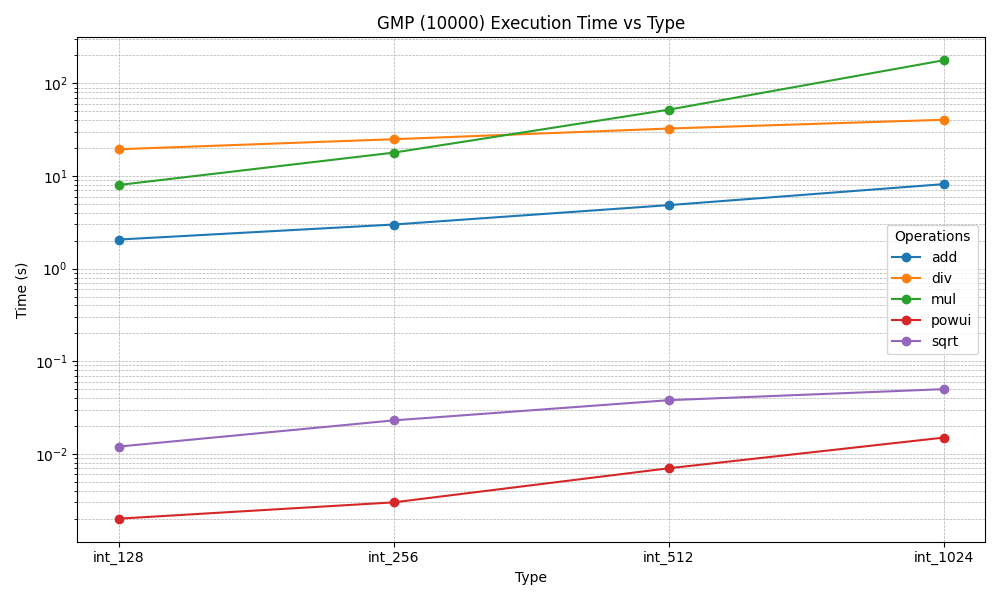


Рисунок 1. Время выполнения GMP операций в зависимости от разрядности целого числа

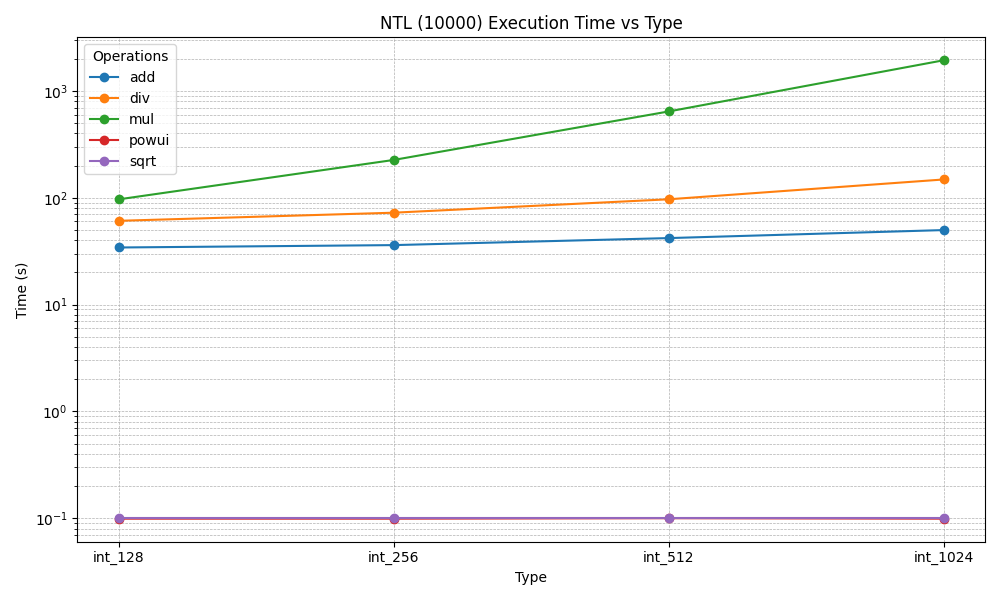


Рисунок 2. Время выполнения NTL операций в зависимости от разрядности целого числа

Как и следовало ожидать, время выполнения операции увеличивается с увеличением разрядности числа. Так же, можно заметить, что самой тяжелой операцией для целочисленных типов является умножение, как для GMP, так и для NTL. Однако, для разрядностей 128, 256 бит целого числа в библиотеке GMP операция деления выполняется дольше, чем умножение.

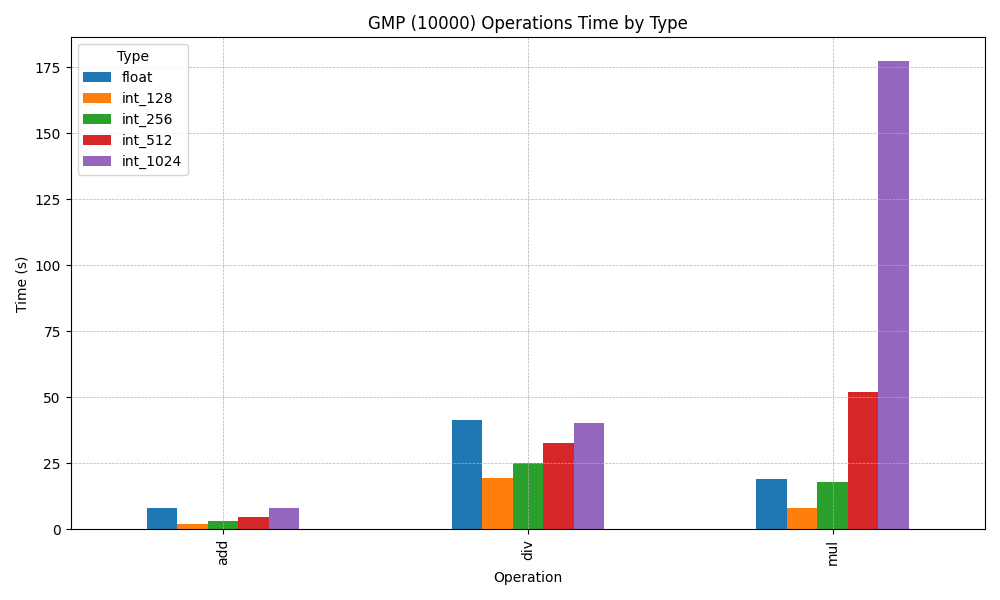


Рисунок 3. График времени выполнения бинарных GMP операций

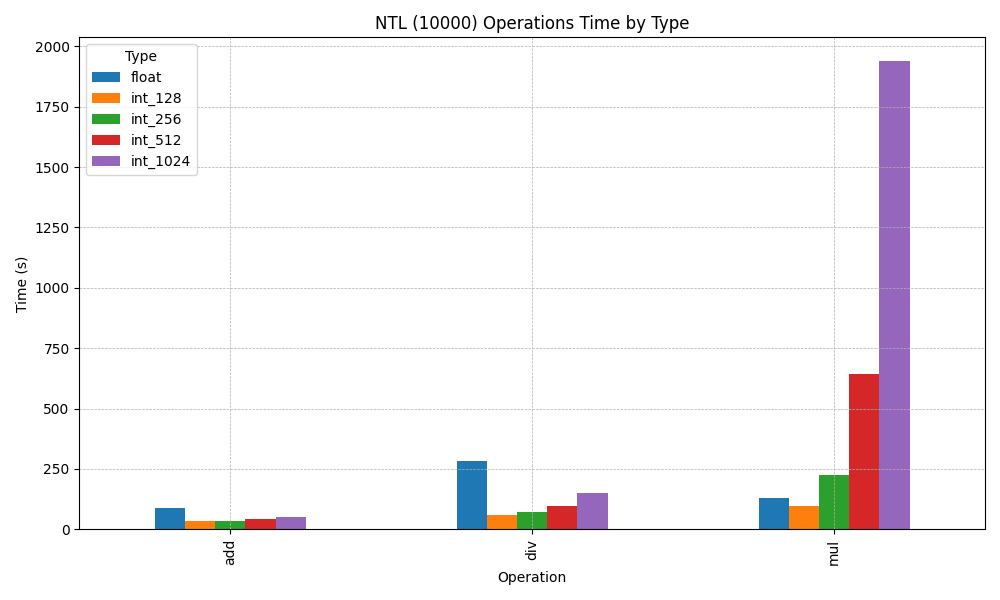


Рисунок 4. График времени выполнения бинарных NTL операций

Из рисунков 3, 4 можно заметить, что характер времени выполнения операций для разных библиотек одинаков. Внутри каждой из библиотек деление является самой тяжелой операцией для вещественных чисел, а время умножения целых чисел разрядности 1024 сильно выше, чем для других операций, разрядностей.

Характер выполнения унарных операций обсуждался ранее, графическое представление смысла не имеет.

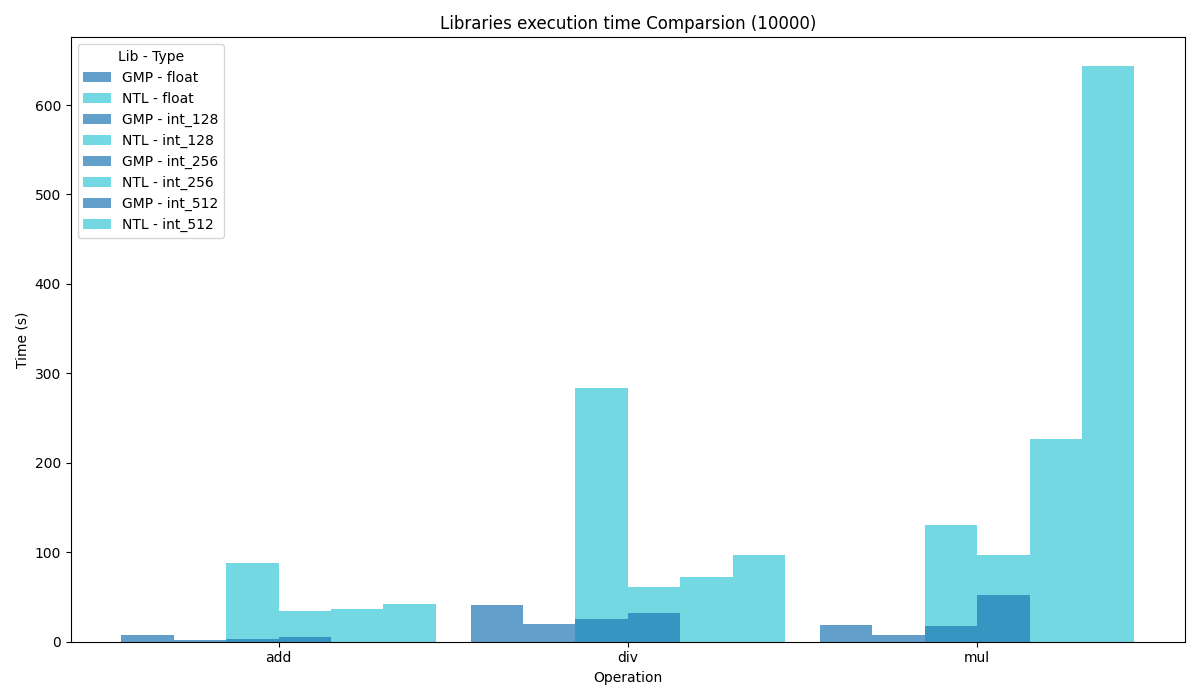


Рисунок 5. Сравнение производительности GMP, NTL

Из рисунка 5 очевидно, что библиотека NTL в разы медленнее библиотеки GMP, как и обсуждалось ранее. Причинами данного расхождения являются алгоритмические различия, различия в уровнях абстракции. В библиотеке GMP используются более оптимизированные алгоритмы для работы с большими числами, например Алгоритм Штрассена, в то время как NTL использует более простые алгоритмы, например алгоритм длинной арифметики. Немаловажным является уровни абстракции, реализованные в библиотеках – GMP близка к архитектуре и ориентирована на низкоуровневые операции, в то время как NTL предоставляет более высокоуровневый интерфейс для работы с числами, добавляя сложные абстракции, которые делают код более понятным и простым в работе, однако накладывают некоторые ресурсные расходы на операции.

**Вывод**

В ходе данной работы были рассмотрены библиотеки GMP и NTL. Сравнение производительности показало, что NTL менее производительна, однако её использование проще, потому как типы, представленные в библиотеке реализованы при помощи классов с соответствующими перегрузками операторов, механизмами инициализации, удаления используемой памяти.

**ПРИЛОЖЕНИЕ А. АЛГОРИТМ ТЕСТИРОВАНИЯ C++.**

// gmptest.h

#pragma once

#include <gmp.h>

static void mpz\_root\_ui(mpz\_ptr rop, mpz\_srcptr op, mpir\_ui n) { mpz\_root(rop, op, n); }

static mpir\_ui POWM\_DEFAULT = 2U;

typedef void (\*bmpz\_func\_t)(mpz\_ptr, mpz\_srcptr, mpz\_srcptr);

typedef void (\*bmpz\_ui\_func\_t)(mpz\_ptr, mpz\_srcptr, mpir\_ui);

typedef void (\*bmpf\_func\_t)(mpf\_ptr, mpf\_srcptr, mpf\_srcptr);

typedef void (\*bmpf\_ui\_func\_t)(mpf\_ptr, mpf\_srcptr, mpir\_ui);

void init\_rand(mpz\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate);

void init\_rand(mpf\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate);

void clear\_array(mpz\_t\* array, size\_t size);

void clear\_array(mpf\_t\* array, size\_t size);

double gmp\_test(mpz\_t\* array, size\_t size, bmpz\_func\_t func);

double gmp\_test(mpz\_t\* array, size\_t size, bmpz\_ui\_func\_t func, mpir\_ui op);

double gmp\_test(mpf\_t\* array, size\_t size, bmpf\_func\_t func);

double gmp\_test(mpf\_t\* array, size\_t size, bmpf\_ui\_func\_t func, mpir\_ui op);

double gmp\_test(mpf\_t\* array, size\_t size); // sqrt

void inttest(FILE\* fout, mpz\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate);

void floattest(FILE\* fout, mpf\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate);

void gmp\_testprogram(FILE\* fout, size\_t size);

// gmptest.cpp

#include "gmptest.h"

void init\_rand(mpz\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate)

{

for (size\_t i = 0; i < size; i++)

{

mpz\_init(array[i]);

mpz\_rrandomb(array[i], rstate, bits);

}

}

void init\_rand(mpf\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate)

{

mpf\_set\_default\_prec(10);

for (size\_t i = 0; i < size; i++)

{

mpf\_init(array[i]);

mpf\_urandomb(array[i], rstate, bits);

}

}

void clear\_array(mpz\_t\* array, size\_t size)

{

for (size\_t i = 0; i < size; i++)

mpz\_clear(array[i]);

}

void clear\_array(mpf\_t\* array, size\_t size)

{

for (size\_t i = 0; i < size; i++)

mpf\_clear(array[i]);

}

double gmp\_test(mpz\_t\* array, size\_t size, bmpz\_func\_t func)

{

mpz\_t result;

mpz\_init(result);

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

for (size\_t j = 0; j < size; j++)

func(result, array[i], array[j]);

clock\_t clend = clock();

mpz\_clear(result);

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

double gmp\_test(mpz\_t\* array, size\_t size, bmpz\_ui\_func\_t func, mpir\_ui op)

{

mpz\_t result;

mpz\_init(result);

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

func(result, array[i], op);

clock\_t clend = clock();

mpz\_clear(result);

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

double gmp\_test(mpf\_t\* array, size\_t size, bmpf\_func\_t func)

{

mpf\_t result;

mpf\_init(result);

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

for (size\_t j = 0; j < size; j++)

func(result, array[i], array[j]);

clock\_t clend = clock();

mpf\_clear(result);

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

double gmp\_test(mpf\_t\* array, size\_t size, bmpf\_ui\_func\_t func, mpir\_ui op)

{

mpf\_t result;

mpf\_init(result);

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

func(result, array[i], op);

clock\_t clend = clock();

mpf\_clear(result);

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

double gmp\_test(mpf\_t\* array, size\_t size)

{

mpf\_t result;

mpf\_init(result);

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

mpf\_sqrt(result, array[i]);

clock\_t clend = clock();

mpf\_clear(result);

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

void inttest(FILE\* fout, mpz\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate)

{

mpir\_ui argui(POWM\_DEFAULT);

fprintf\_s(fout, "TEST: int\nSIZE: %zu\nBITS: %zu\nRESULTS: {\n", size, bits);

init\_rand(array, size, bits, rstate);

fprintf\_s(fout, " add: %.3fs\n", gmp\_test(array, size, mpz\_add));

fprintf\_s(fout, " mul: %.3fs\n", gmp\_test(array, size, mpz\_mul));

fprintf\_s(fout, " div: %.3fs\n", gmp\_test(array, size, mpz\_div));

fprintf\_s(fout, " powui: %.3fs\n", gmp\_test(array, size, mpz\_pow\_ui, argui));

fprintf\_s(fout, " rootui: %.3fs\n}\n", gmp\_test(array, size, mpz\_root\_ui, argui));

clear\_array(array, size);

}

void floattest(FILE\* fout, mpf\_t\* array, size\_t size, mp\_bitcnt\_t bits, gmp\_randstate\_t rstate)

{

mpir\_ui argui(POWM\_DEFAULT);

fprintf\_s(fout, "TEST: float\nSIZE: %zu\nRESULTS: {\n", size);

init\_rand(array, size, bits, rstate);

fprintf\_s(fout, " add: %.3fs\n", gmp\_test(array, size, mpf\_add));

fprintf\_s(fout, " mul: %.3fs\n", gmp\_test(array, size, mpf\_mul));

fprintf\_s(fout, " div: %.3fs\n", gmp\_test(array, size, mpf\_div));

fprintf\_s(fout, " powui: %.3fs\n", gmp\_test(array, size, mpf\_pow\_ui, argui));

fprintf\_s(fout, " sqrt: %.3fs\n}\n", gmp\_test(array, size));

clear\_array(array, size);

}

void gmp\_testprogram(FILE\* fout, size\_t size)

{

gmp\_randstate\_t rstate;

gmp\_randinit\_default(rstate);

mpz\_t\* zarray = new mpz\_t[size];

mpf\_t\* farray = new mpf\_t[size];

inttest(fout, zarray, size, 128U, rstate);

inttest(fout, zarray, size, 256U, rstate);

inttest(fout, zarray, size, 512U, rstate);

inttest(fout, zarray, size, 1024U, rstate);

floattest(fout, farray, size, 1024U, rstate);

delete[] zarray;

delete[] farray;

}

// ntltest.h

#pragma once

#pragma warning(disable:4146)

#include <NTL/ZZ.h>

#include <NTL/RR.h>

using namespace NTL;

static uint64\_t POWM\_DEFAULT = 2U;

enum class op\_t

{

add,

mul,

div,

pow,

root

};

void init\_rand(ZZ\* array, size\_t size, long bits);

void init\_rand(RR\* array, size\_t size);

double ntl\_test(ZZ\* array, size\_t size, op\_t f);

double ntl\_test(ZZ\* array, size\_t size, op\_t f, long op);

double ntl\_test(RR\* array, size\_t size, op\_t f);

double ntl\_test(RR\* array, size\_t size, op\_t f, long op);

void inttest(FILE\* fout, ZZ\* array, size\_t size, long bits);

void floattest(FILE\* fout, RR\* array, size\_t size);

void ntl\_testprogram(FILE\* fout, size\_t size);

// ntltest.cpp

#include "ntltest.h"

void init\_rand(ZZ\* array, size\_t size, long bits)

{

for (int i = 0; i < size; i++)

{

array[i] = RandomBits\_ZZ(bits);

}

}

void init\_rand(RR\* array, size\_t size)

{

for (int i = 0; i < size; i++)

{

array[i] = random\_RR() \* 100000000;

}

}

double ntl\_test(ZZ\* array, size\_t size, op\_t f)

{

ZZ result;

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

for (size\_t j = 0; j < size; j++)

switch (f)

{

case op\_t::add:

result = array[i] + array[j];

break;

case op\_t::mul:

result = array[i] \* array[j];

break;

case op\_t::div:

result = array[i] / array[j];

break;

default:

break;

}

clock\_t clend = clock();

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

double ntl\_test(ZZ\* array, size\_t size, op\_t f, long op)

{

ZZ result;

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

switch (f)

{

case op\_t::pow:

result = power(array[i], op);

break;

case op\_t::root:

result = SqrRoot(array[i]);

break;

default:

break;

}

clock\_t clend = clock();

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

double ntl\_test(RR\* array, size\_t size, op\_t f)

{

RR result;

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

for (size\_t j = 0; j < size; j++)

switch (f)

{

case op\_t::add:

result = array[i] + array[j];

break;

case op\_t::mul:

result = array[i] \* array[j];

break;

case op\_t::div:

result = array[i] / array[j];

break;

default:

break;

}

clock\_t clend = clock();

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

double ntl\_test(RR\* array, size\_t size, op\_t f, long op)

{

RR result;

clock\_t clbeg = clock();

for (size\_t i = 0; i < size; i++)

switch (f)

{

case op\_t::pow:

result = power(array[i], op);

break;

case op\_t::root:

result = SqrRoot(array[i]);

break;

default:

break;

}

clock\_t clend = clock();

return (clend - clbeg) / (double)CLOCKS\_PER\_SEC;

}

void inttest(FILE\* fout, ZZ\* array, size\_t size, long bits)

{

fprintf\_s(fout, "TEST: int\nSIZE: %zu\nBITS: %d\nRESULTS: {\n", size, bits);

init\_rand(array, size, bits);

fprintf\_s(fout, " add: %.3fs\n", ntl\_test(array, size, op\_t::add));

fprintf\_s(fout, " mul: %.3fs\n", ntl\_test(array, size, op\_t::mul));

fprintf\_s(fout, " div: %.3fs\n", ntl\_test(array, size, op\_t::div));

fprintf\_s(fout, " powui: %.3fs\n", ntl\_test(array, size, op\_t::pow));

fprintf\_s(fout, " sqrt: %.3fs\n}\n", ntl\_test(array, size, op\_t::root));

}

void floattest(FILE\* fout, RR\* array, size\_t size)

{

fprintf\_s(fout, "TEST: float\nSIZE: %zu\nRESULTS: {\n", size);

init\_rand(array, size);

fprintf\_s(fout, " add: %.3fs\n", ntl\_test(array, size, op\_t::add));

fprintf\_s(fout, " mul: %.3fs\n", ntl\_test(array, size, op\_t::mul));

fprintf\_s(fout, " div: %.3fs\n", ntl\_test(array, size, op\_t::div));

fprintf\_s(fout, " powui: %.3fs\n", ntl\_test(array, size, op\_t::pow));

fprintf\_s(fout, " sqrt: %.3fs\n}\n", ntl\_test(array, size, op\_t::root));

}

void ntl\_testprogram(FILE\* fout, size\_t size)

{

ZZ\* zarray = new ZZ[size];

RR\* farray = new RR[size];

inttest(fout, zarray, size, 128);

inttest(fout, zarray, size, 256);

inttest(fout, zarray, size, 512);

inttest(fout, zarray, size, 1024);

floattest(fout, farray, size);

delete[] zarray;

delete[] farray;

}

// common.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdint>

#include <cstdlib>

#include <cstdio>

#include <ctime>

#define IS\_GMP\_TEST 0

#if IS\_GMP\_TEST

#include "gmptest.h"

#else

#include "ntltest.h"

#endif

// main.cpp

#include "common.h"

#define PATH\_FLAG 'p'

#define SIZE\_FLAG 'n'

#define POWM\_FLAG 'w'

void fatal\_exit(int code)

{

fprintf\_s(stderr, "Fatal exit code: %d\n", code);

exit(code);

}

int main(int argc, char\* argv[])

{

FILE\* fout = stdout;

size\_t arrsize = 1000;

for (int i = 1; i < argc; i++)

{

if (argv[i][0] == '-')

switch (argv[i][1])

{

case PATH\_FLAG:

fout = fopen(argv[i + 1], "w+");

if (fout == NULL) {

fprintf\_s(stderr, "Cannot open the file!\n");

fatal\_exit(1);

}

break;

case SIZE\_FLAG:

arrsize = strtoull(argv[i + 1], NULL, 10);

break;

case POWM\_FLAG:

POWM\_DEFAULT = strtoull(argv[i + 1], NULL, 10);

break;

default:

fprintf\_s(stderr, "Undefined flag \"%c\"!\n", argv[i][1]);

fatal\_exit(2);

break;

}

}

clock\_t global\_clbeg = clock();

#if IS\_GMP\_TEST

fprintf\_s(fout, "GMP Lib Test Prorgam\n\n");

gmp\_testprogram(fout, arrsize);

#else

fprintf\_s(fout, "NTL Lib Test Prorgam\n\n");

ntl\_testprogram(fout, arrsize);

#endif

clock\_t global\_clend = clock();

double elapsed = (global\_clend - global\_clbeg) / (double)CLOCKS\_PER\_SEC;

fprintf\_s(fout, "\nTest time: %.3fs\n", elapsed);

if (fout != stdout && fout != NULL)

fclose(fout);

return 0;

}

**ПРИЛОЖЕНИЕ Б. СКРИПТ ИНТЕРПРЕТАЦИИ PYTHON.**

import re

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

img\_path: str = '.\\images\\'

tab\_path: str = '.\\tables\\'

res\_pathes: dict = {

'GMP': '.\\gmp\_result\_',

'NTL': '.\\ntl\_result\_'

}

res\_tests: dict = {

1000: '1k.txt',

10000: '10k.txt'

}

class data\_struct:

def \_\_init\_\_(self, lib: str, size: int, data: pd.DataFrame):

self.name: str = lib

self.size: int = size

self.data: pd.DataFrame = data

def plot\_operations(self, exclude\_types: list[str] = None, exclude\_operations: list[str] = None, time\_unit = 's'):

df\_pivot = self.data.copy()

if exclude\_types is not None:

df\_pivot = df\_pivot.drop(columns=[ty for ty in exclude\_types if ty in df\_pivot.columns])

if exclude\_operations is not None:

df\_pivot = df\_pivot.drop(index=[op for op in exclude\_operations if op in df\_pivot.index])

if time\_unit == 'ms':

df\_pivot = df\_pivot \* 1000

ylabel = 'Time (ms)'

elif time\_unit == 'us':

df\_pivot = df\_pivot \* 1\_000\_000

ylabel = 'Time (µs)'

else:

ylabel = 'Time (s)'

title = f"{self.name} ({self.size}) Operations Time by Type"

ax = df\_pivot.plot(kind='bar', figsize=(10, 6))

ax.set\_title(title)

ax.set\_ylabel(ylabel)

ax.set\_xlabel("Operation")

ax.legend(title="Type")

ax.grid(True, which='both', axis='both', linestyle='--', linewidth=0.5)

plt.tight\_layout()

plt.savefig(img\_path + title.replace(' ', '\_') +

str(len(exclude\_types) if exclude\_types is not None else 0) +

str(len(exclude\_operations) if exclude\_operations is not None else 0))

def plot\_time\_vs\_bits(self, exclude\_types: list[str] = None, exclude\_operations: list[str] = None, time\_unit = 's'):

plt.figure(figsize=(10, 6))

df\_pivot = self.data.drop(columns=['float'])

if exclude\_types is not None:

df\_pivot = df\_pivot.drop(columns=[ty for ty in exclude\_types if ty in df\_pivot.columns])

if exclude\_operations is not None:

df\_pivot = df\_pivot.drop(index=[op for op in exclude\_operations if op in df\_pivot.index])

if time\_unit == 'ms':

df\_pivot = df\_pivot \* 1000

ylabel = 'Time (ms)'

elif time\_unit == 'us':

df\_pivot = df\_pivot \* 1\_000\_000

ylabel = 'Time (µs)'

else:

ylabel = 'Time (s)'

title = f"{self.name} ({self.size}) Execution Time vs Type"

for operation in df\_pivot.index:

operation\_data = df\_pivot.loc[operation]

plt.plot(operation\_data.index, operation\_data.values, marker='o', label=operation)

plt.legend(title="Operations")

plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.5)

plt.title(title)

plt.xlabel("Type")

plt.ylabel(ylabel)

plt.yscale('log')

plt.tight\_layout()

plt.savefig(img\_path + title.replace(' ', '\_') +

str(len(exclude\_types) if exclude\_types is not None else 0) +

str(len(exclude\_operations) if exclude\_operations is not None else 0))

def parse\_results(lib: str, file\_path: str) -> data\_struct:

with open(file\_path, 'r') as file:

content = file.read()

test\_pattern = re.compile(r"TEST:\s\*(\w+)")

size\_pattern = re.compile(r"SIZE:\s\*(\d+)")

bits\_pattern = re.compile(r"BITS:\s\*(\d+)")

results\_pattern = re.compile(r"RESULTS:\s\*\{(.\*?)\}", re.DOTALL)

tests = test\_pattern.findall(content)

sizes = size\_pattern.findall(content)

bits = bits\_pattern.findall(content)

results\_blocks = results\_pattern.findall(content)

operations\_data = []

for i, block in enumerate(results\_blocks):

operations = re.findall(r"(\w+):\s\*([\d.]+)s", block)

for operation, time in operations:

operations\_data.append({

"Type": 'int\_' + bits[i] if i < len(bits) else 'float',

"Operation": operation,

"Time\_s": float(time)

})

df\_operations = pd.DataFrame(operations\_data)

df\_pivot = df\_operations.pivot(index='Operation', columns='Type', values='Time\_s')

df\_pivot = df\_pivot[['float', 'int\_128', 'int\_256', 'int\_512', 'int\_1024']]

return data\_struct(lib, sizes[0], df\_pivot)

def plot\_comparsion(dfs: list[pd.DataFrame], names: list[str], size: int, exclude\_types: list[str] = None, exclude\_operations: list[str] = None, time\_unit = 's'):

if len(dfs) != len(names):

raise ValueError("Size of 'dfs', 'names' mismatch!")

for i, df\_pivot in enumerate(dfs):

if exclude\_types is not None:

df\_pivot = df\_pivot.drop(columns=[ty for ty in exclude\_types if ty in df\_pivot.columns])

if exclude\_operations is not None:

df\_pivot = df\_pivot.drop(index=[op for op in exclude\_operations if op in df\_pivot.index])

dfs[i] = df\_pivot

if time\_unit == 'ms':

df\_pivot = df\_pivot \* 1000

ylabel = 'Time (ms)'

elif time\_unit == 'us':

df\_pivot = df\_pivot \* 1\_000\_000

ylabel = 'Time (µs)'

else:

ylabel = 'Time (s)'

all\_data = []

for table, library in zip(dfs, names):

table['library'] = library

table.reset\_index(inplace=True)

all\_data.append(table)

df\_pivot = pd.concat(all\_data, ignore\_index=True)

operations = df\_pivot['Operation'].unique()

num\_operations = len(operations)

bar\_width = 0.15

index = np.arange(num\_operations)

colors = plt.get\_cmap('tab10', len(names))

fig, ax = plt.subplots(figsize=(12, 7))

for i, col in enumerate(df\_pivot.columns[1:-2]):

for j, library in enumerate(names):

library\_data = df\_pivot[df\_pivot['library'] == library]

alpha = 0.7 - (j \* 0.1)

ax.bar(index + i \* bar\_width + j \* bar\_width \* len(names),

library\_data[col],

bar\_width,

label=f"{library} - {col}",

color=colors(j),

alpha=alpha)

title = f'Libraries execution time Comparsion ({size})'

ax.set\_xlabel('Operation')

ax.set\_ylabel(ylabel)

ax.set\_title(title)

ax.set\_xticks(index + bar\_width \* (len(names) - 1) / 2 + 0.3)

ax.set\_xticklabels(operations)

ax.legend(title='Lib - Type')

plt.tight\_layout()

plt.savefig(img\_path + title.replace(' ', '\_') +

str(len(exclude\_types) if exclude\_types is not None else 0) +

str(len(exclude\_operations) if exclude\_operations is not None else 0))

if \_\_name\_\_ == '\_\_main\_\_':

results: dict[dict[data\_struct]] = {}

for lib, path in res\_pathes.items():

results[lib] = dict()

for size, test in res\_tests.items():

results[lib][size] = parse\_results(lib, path + test)

operations = ['add', 'mul', 'div', 'powui', 'sqrt']

for lib, result in results.items():

for size, test in result.items():

test.plot\_operations(exclude\_operations=operations[3:])

test.plot\_operations(exclude\_operations=operations[:3])

test.plot\_time\_vs\_bits()

test.data.to\_csv(tab\_path + f'{lib}\_{size}.csv', sep=';', index=True)

plot\_comparsion([

results['GMP'][10000].data,

results['NTL'][10000].data,

], ['GMP', 'NTL'], 10000, exclude\_operations=operations[:3])

plot\_comparsion([

results['GMP'][10000].data,

results['NTL'][10000].data,

], ['GMP', 'NTL'], 10000, exclude\_operations=operations[3:])