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 C++.

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

1. Обзор библиотек

1.1. 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:

$$mpz\_add(mpz\_t\ rop, mpz\_t\ op1, mpz\_t\ op2)$$

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.

1.2. 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.

2. 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 $\sqrt[n]{a}$, 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.

3. 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:

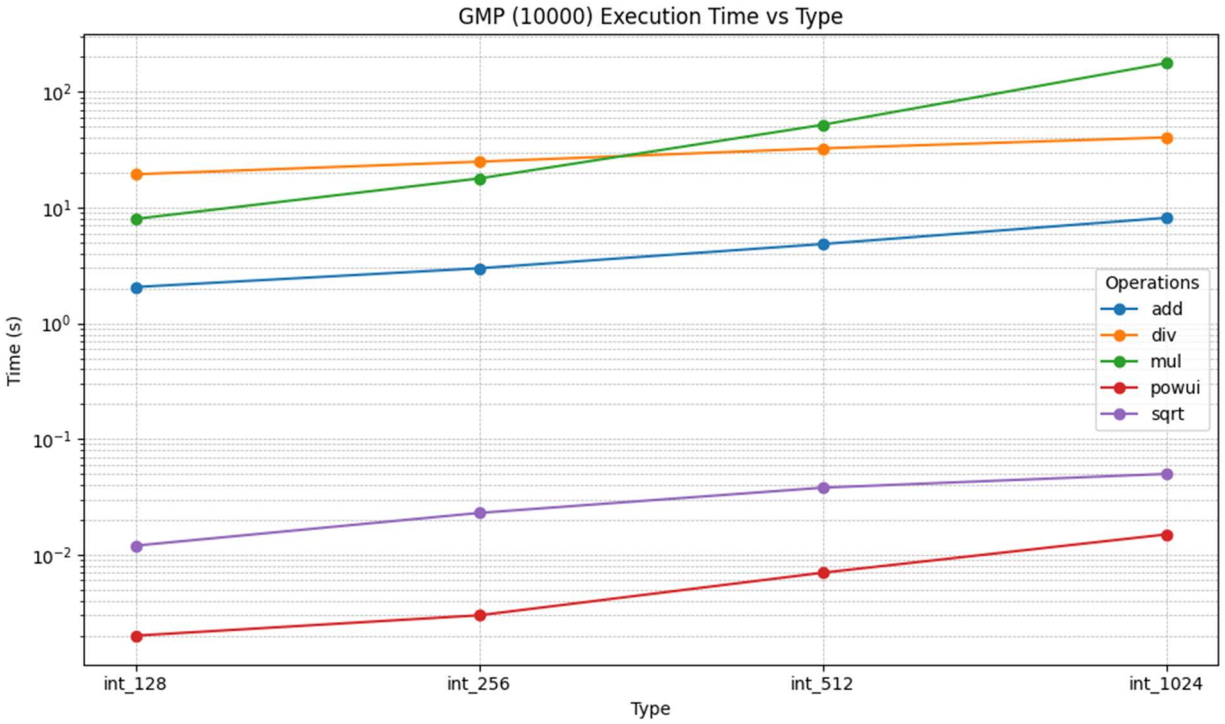

Рисунок 1. Время выполнения GMP операций в зависимости от разрядности целого числа

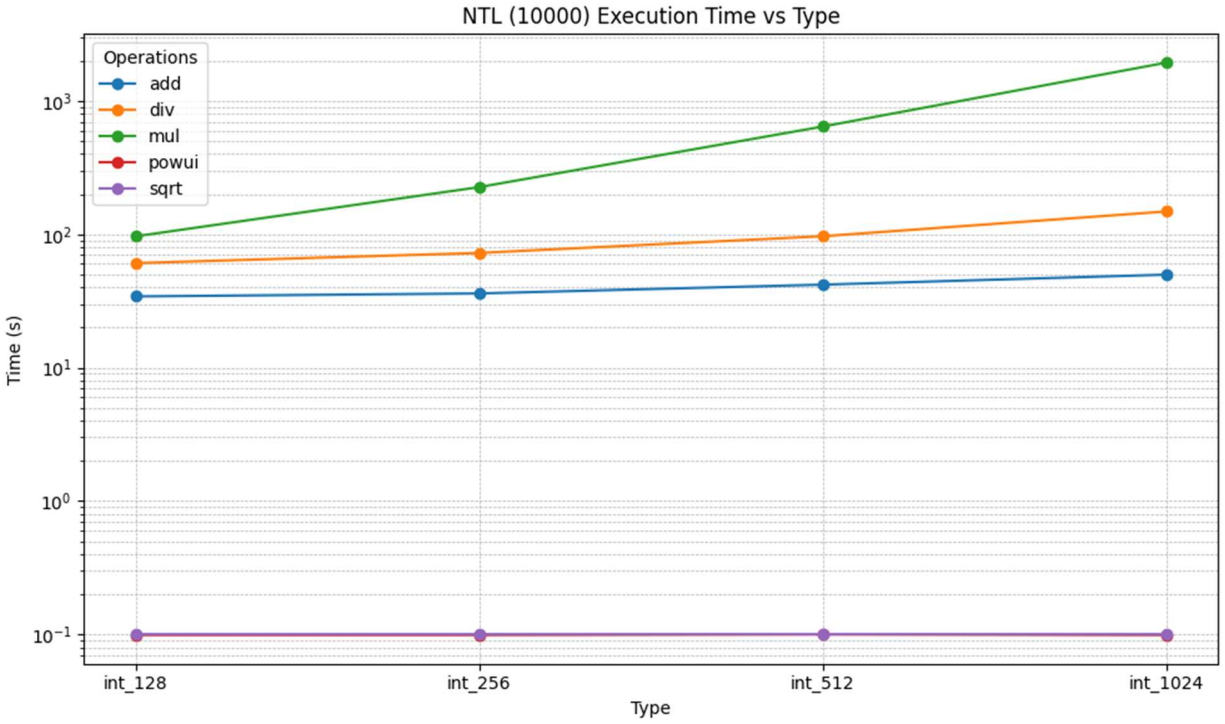

Рисунок 2. Время выполнения NTL операций в зависимости от разрядности целого числа

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.

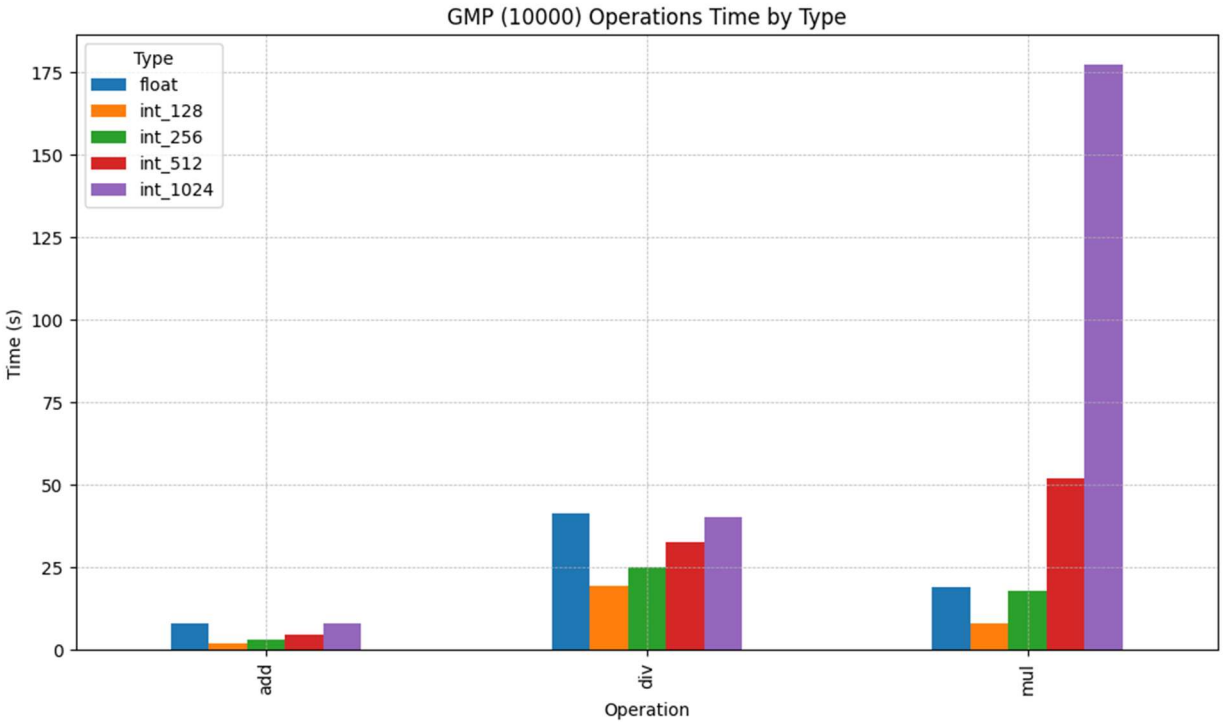

Рисунок 3. График времени выполнения бинарных GMP операций

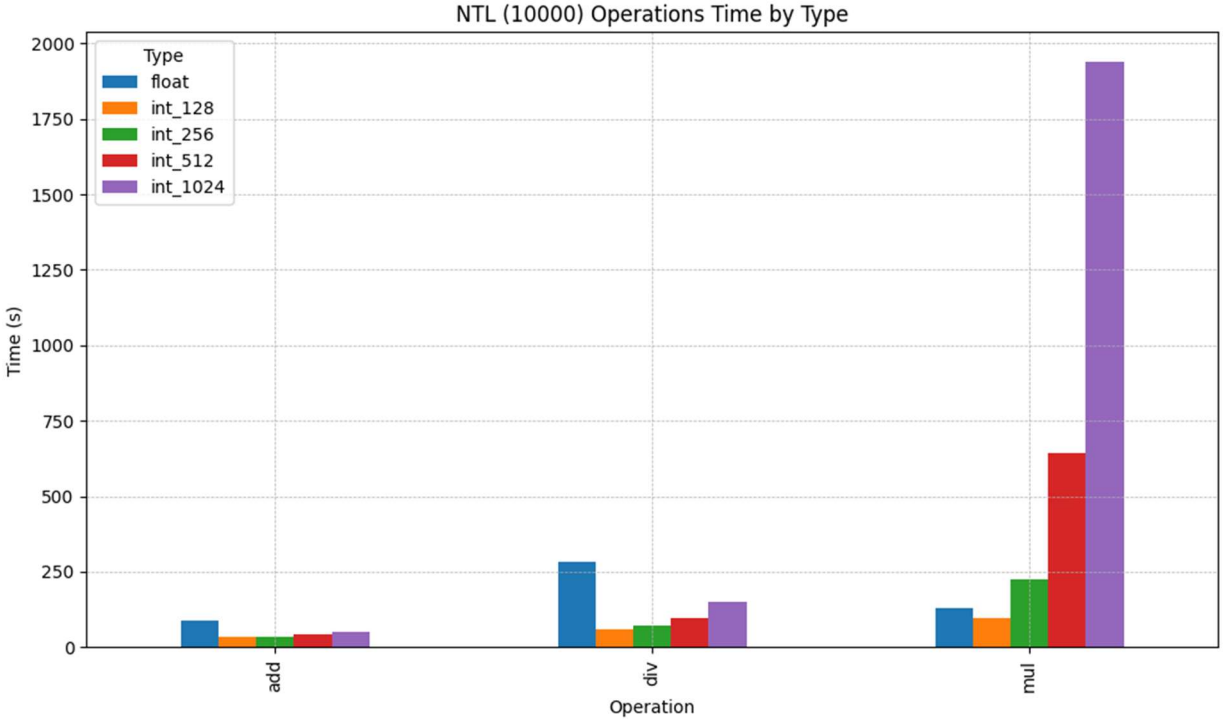

Рисунок 4. График времени выполнения бинарных NTL операций

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.

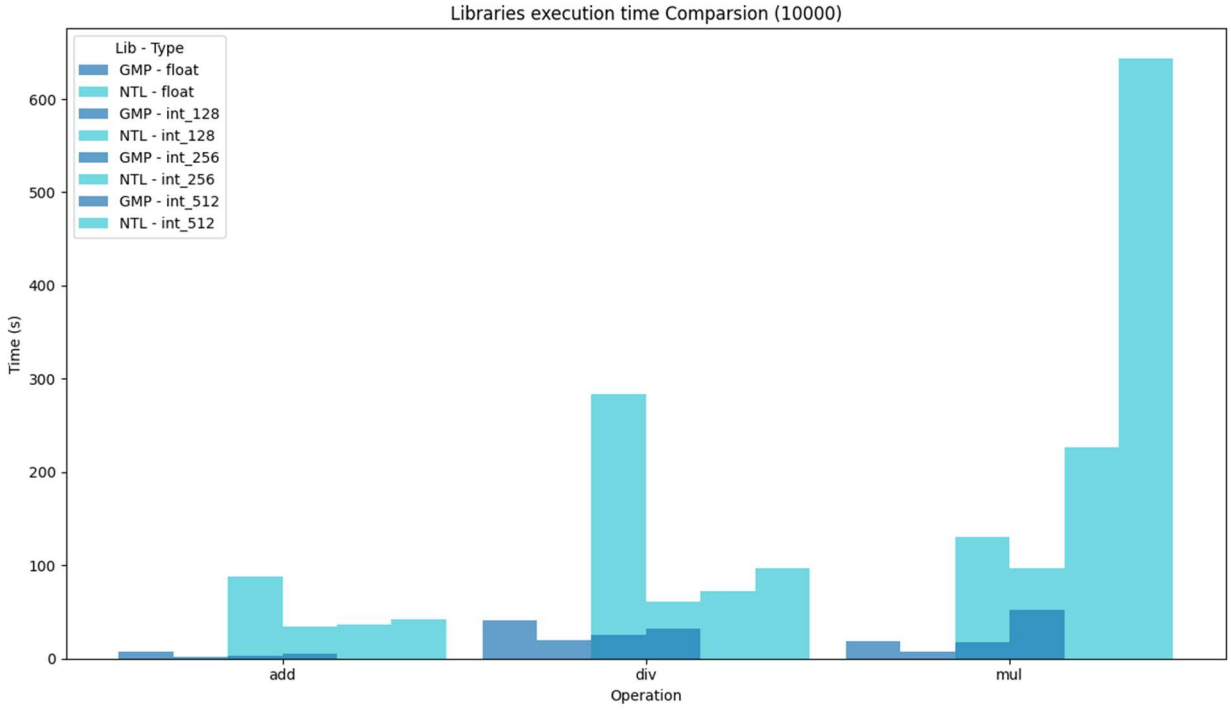

Рисунок 5. Сравнение производительности GMP, NTL

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:])
```