Московский Государственный Университет им. М.В. Ломоносова

Факультет Вычислительной Математики и Кибернетики

Кафедра Суперкомпьютеров и Квантовой Информатики

Практикум на ЭВМ
Отчёт № 3
Параллельная программа на MPI и OpenMP, реализующая однокубитное квантовое преобразование с шумами

Работу выполнил

Сайбель Т. А.

Москва 2021

Описание функций, типов и переменных библиотеки quantum.h

complexd ~ std::complex<double>

complexd *read(const char *f, int *n)
Читает вектор состояний из файла file.

void write(const char *f, const complexd *A, unsigned int n)
Записывает вектор состояний A для n кубитов в файл file.

int convert(const complexd *A, complexd *B, unsigned int n, unsigned int i, complexd **P, const unsigned int *k, complexd *BUF = nullptr)
Производит i-кубитное квантовое преобразование вектора состояний A для n кубитов квантовым вентилем P (указатель на двумерный complexd массив размера $2^i \times 2^i$) над кубитами k (указатель на unsigned int массив размера i). Записывает полученный вектор (участок для каждого процесса) в B. Позволяет для оптимизации напрямую указывать буфер BUF для хранения данных, присылаемых с других процессов (должен быть не меньше необходимого размера). Возвращает 0, если не возникло ошибок.

complexd *qubitConvert(const complexd *A, unsigned int n, unsigned int i, complexd **P, const unsigned int *k)
Производит i-кубитное квантовое преобразование вектора состояний A для n кубитов квантовым вентилем P (указатель на двумерный complexd массив размера $2^i \times 2^i$) над кубитами k (указатель на unsigned int массив размера i). Возвращает каждому процессу указатель на complexd массив – участок полученного вектора.

complexd *Hadamard(const complexd *A, unsigned int n, unsigned int k)
Производит однокубитное квантовое преобразование Адамара вектора состояний A для n кубитов над кубитом k. Возвращает каждому процессу указатель на complexd массив – участок полученного вектора.

complexd *nHadamard(const complexd *A, unsigned int n)
Производит n-Адамар преобразование вектора состояний A для n кубитов.
Возвращает каждому процессу указатель на complexd массив – участок полученного вектора.

complexd *NOT(const complexd *A, unsigned int n, unsigned int k)
Производит отрицание вектора состояний A для n кубитов над кубитом k. Возвращает каждому процессу указатель на complexd массив – участок полученного вектора.

complexd *CNOT(const complexd *A, unsigned int n, unsigned int k1, unsigned int k2)

Производит контролируемое отрицание вектора состояний A для n кубитов над кубитами k1 и k2. Возвращает каждому процессу указатель на complexd массив – участок полученного вектора.

complexd *ROT(const complexd *A, unsigned int n, unsigned int k, double a)
Производит фазовый сдвиг на угол alfa вектора состояний A для n кубитов над кубитом k. Возвращает каждому процессу указатель на complexd массив – участок полученного вектора.

complexd *CROT(const complexd *A, unsigned int n, unsigned int k1, unsigned int k2, double a)
Производит контролируемый поворот на угол alfa вектора состояний A для n кубитов над кубитами k1 и k2. Возвращает каждому процессу указатель на complexd массив – участок полученного вектора.

void init()
Инициализирует библиотеку. Устанавливает значения переменных, указанных ниже.

int rank	номер MPI процесса
int size	количество MPI процессов (должно быть 2^n)
int logSize	двоичный логарифм от size
int threads	количество OpenMP потоков
bool initFlag	true, если init() уже вызывалась

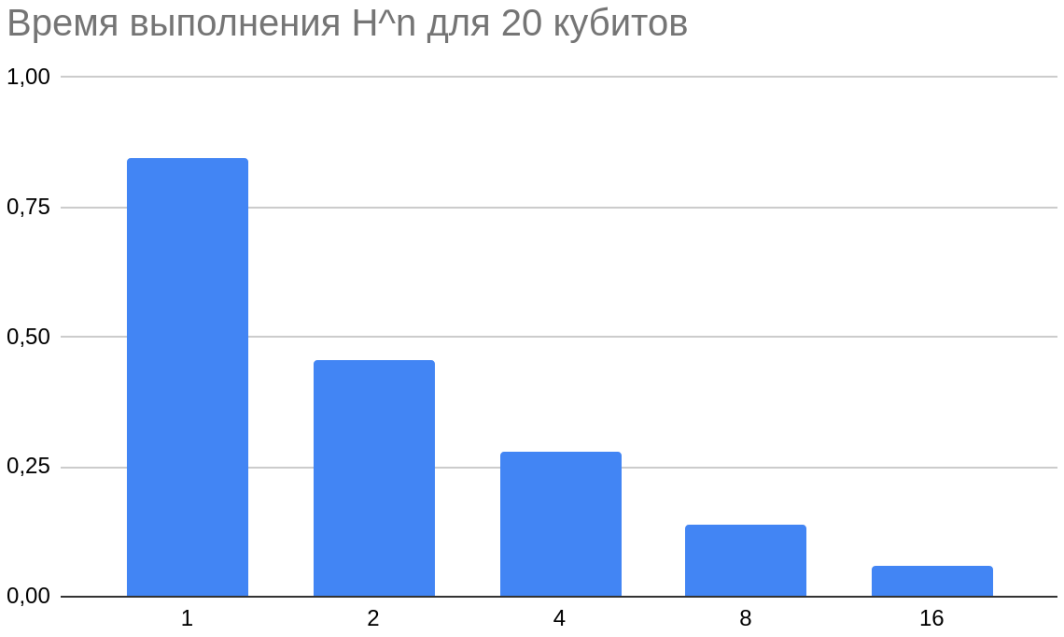
unsigned long long numOfDoubles(int n)
Вычисляет размер участка вектора состояний для n кубитов на процессе (размер массива).

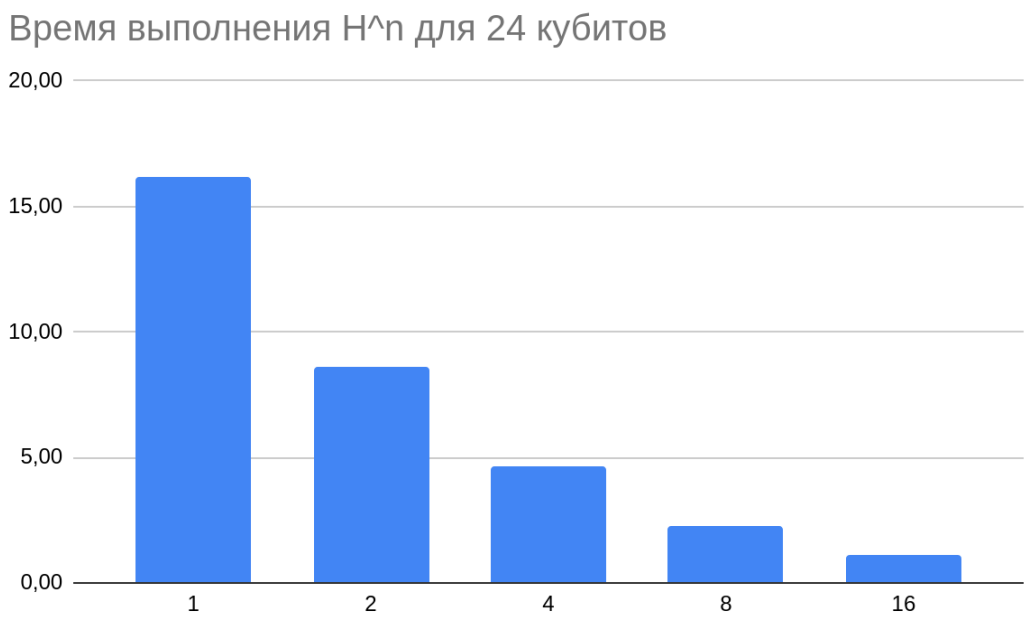
unsigned long long *getMasks(unsigned int i, const unsigned int *k, unsigned int len)
Вычисляет побитовые маски, последовательно переключая биты с номерами из массива k длины i (нумерация начиная с позиции len слева направо). Возвращает указатель на массив размера 2^i.

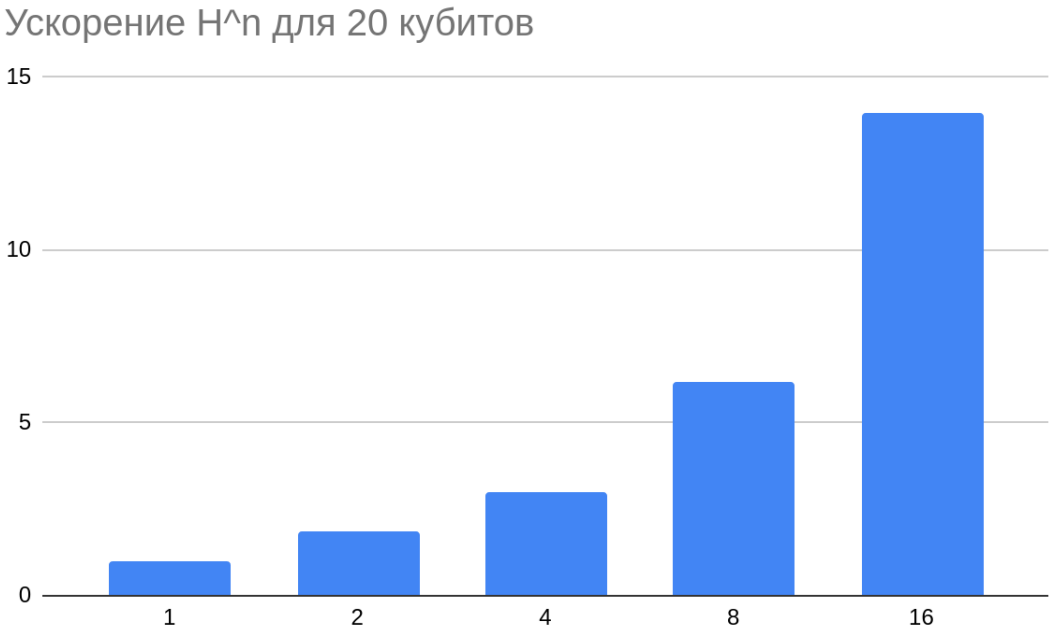
unsigned int *getRanks(unsigned int i, const unsigned int *k)
Вычисляет номера MPI процессов, с которыми текущему процессу необходимо совершить обмен данными, для выполнения i-кубитного квантового преобразования над кубитами k. Возвращает указатель на массив, нулевой элемент которого – количество процессов n,
остальные n элементов – номера этих процессов.

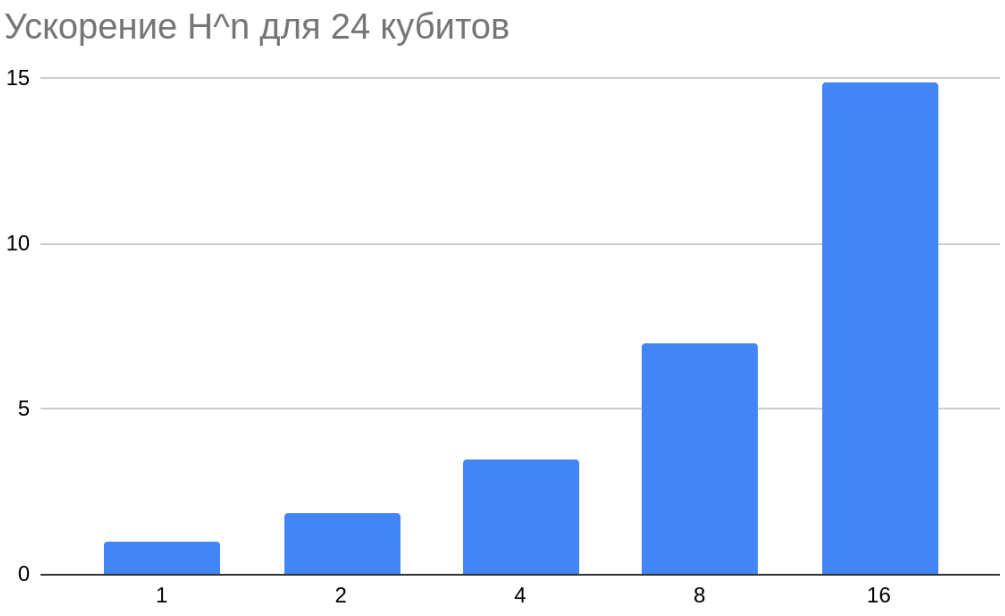
Результаты выполнения H^n

Количество кубитов (n)	Количество вычислительных узлов	Время работы (сек)	Ускорение
20	1	0,85	1
	2	0,456523	1,853939451
	4	0,280772	3,014424515
	8	0,137385	6,160541544
	16	0,06063	13,95952499
24	1	16,14	1
	2	8,61621	1,873317851
	4	4,66105	3,462932172
	8	2,300161925	7,017288576
	16	1,083491237	14,89712094



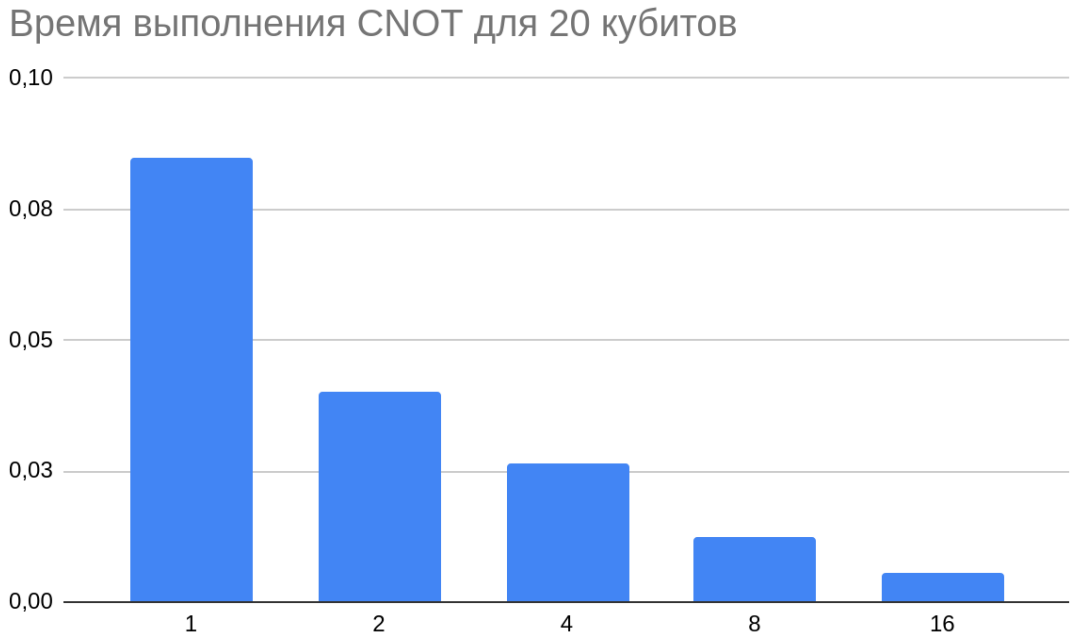

Время выполнения H^n для 24 кубитов
20,00
15,00
10,00
5,00
0,00
1
2
4
8
16


Ускорение H^n для 20 кубитов
15
10
5
0
1
2
4
8
16

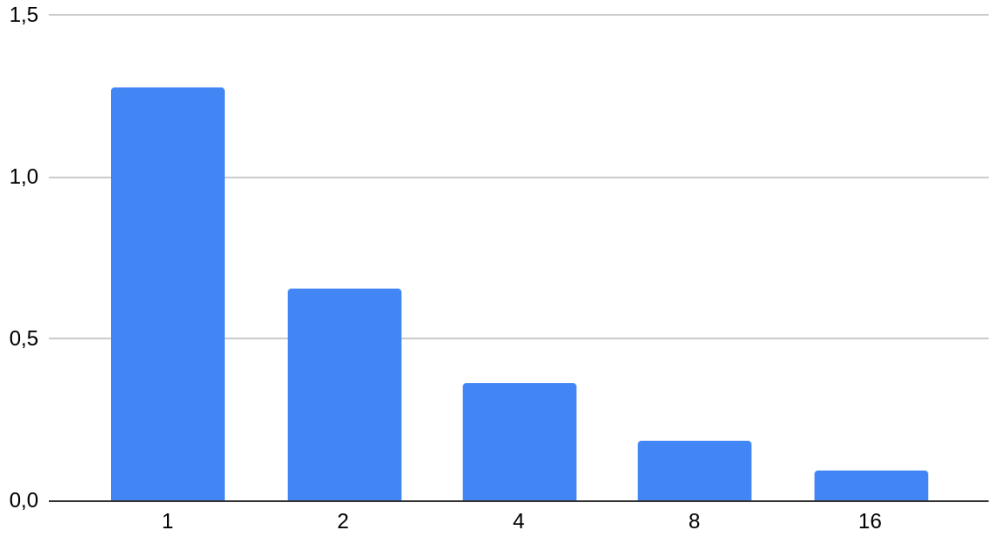

Ускорение H^n для 24 кубитов
15
10
5
0
1
2
4
8
16

Результаты выполнения CNOT

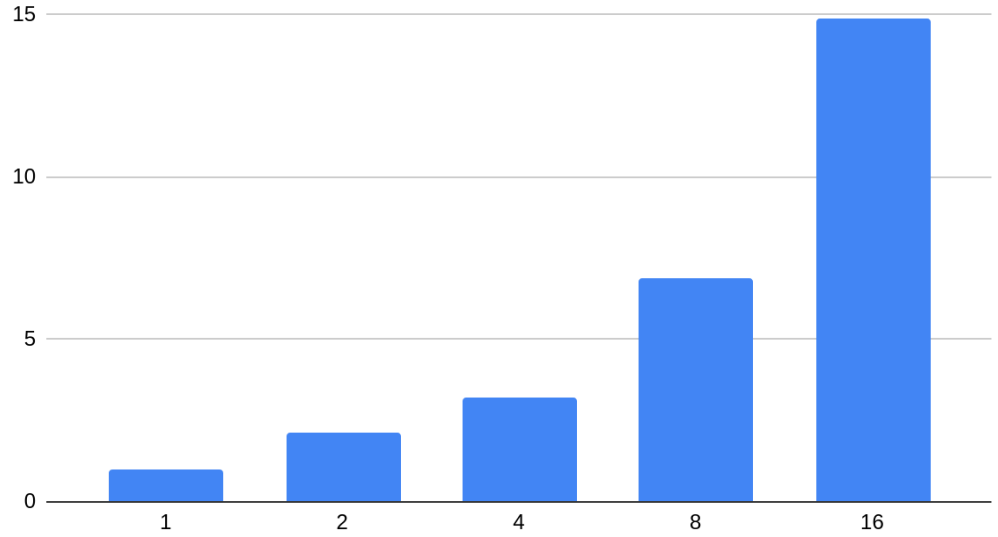
Количество кубитов (n)	Количество вычислительных узлов	Время работы (сек)	Ускорение
20	1	0,08	1
	2	0,0401991	2,113567219
	4	0,02659	3,195317789
	8	0,012311318	6,901251353
	16	0,005699247	14,9078466
24	1	1,2805	1
	2	0,654792	1,955582842
	4	0,362964703	3,527891251
	8	0,184372499	6,945178955
	16	0,092194433	13,88912495



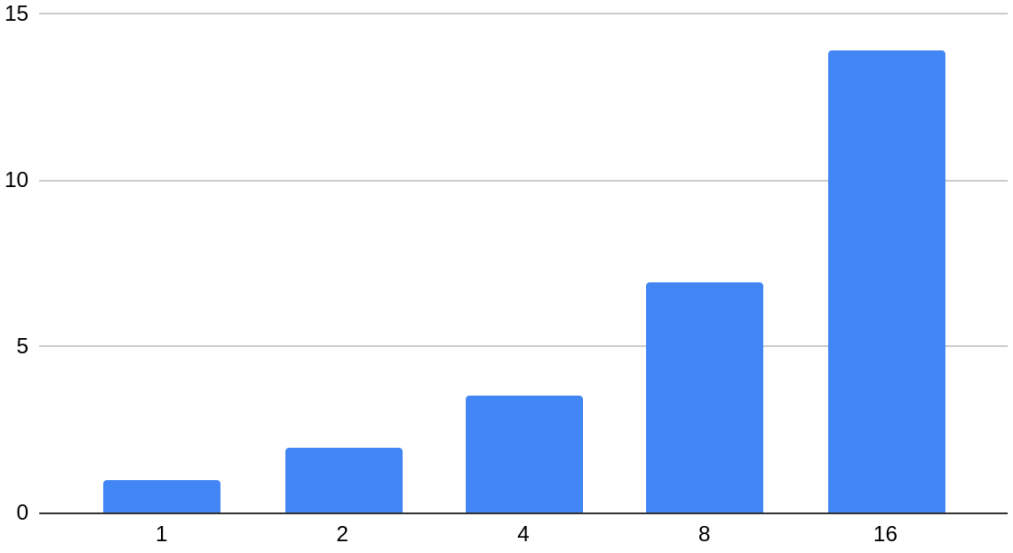
Время выполнения CNOT для 24 кубитов



Ускорение CNOTдля 20 кубитов



Ускорение CNOT для 24 кубитов



Основные выводы

Распараллеливание с использованием технологии MPI ускоряет выполнение программы, но с увеличением числа процессов эффективность снижается из-за роста количества пересылок. Следовательно, можно сделать вывод о хорошей масштабируемости программы.