МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ
КАФЕДРА СУПЕРКОМПЬЮТЕРОВ И КВАНТОВОЙ ИНФОРМАТИКИ

КАФЕДРАЛЬНЫЙ ПРАКТИКУМ

ЗАДАНИЕ 1:
ПАРАЛЛЕЛЬНАЯ ПРОГРАММА НА OPENMP, РЕАЛИЗУЮЩАЯ ОДНОКУБИТНОЕ КВАНТОВОЕ ПРЕОБРАЗОВАНИЕ

Выполнил:
Алёшин Н.А.
группа 323

Москва 2020

Постановка задачи.

Реализовать параллельную программу на $C{+}{+}$ с использованием OpenMP, которая выполняет однокубитное квантовое преобразование над вектором состояний длины 2^n, где n – количество кубитов, по указанному номеру кубита k.

Определить максимальное количество кубитов, для которых возможна работа программы на системе Polus. Выполнить теоретический расчет и проверить его экспериментально.

Начальное состояние вектора должно генерироваться случайным образом.

Протестировать программу на системе Polus. В качестве теста использовать преобразование Адамара по номеру кубита:

a) 1;
b) 2 (номер в списке группы $+ 1$);
c) n.

Подсчет максимального количества кубитов.

На одном узле системы Polus $256\ Gb$ оперативной памяти.

$sizeof(complex < double >) = 16\ b$

Вектор состояний длины 2^{33} ($n = 33$ кубита):

$$2^{33} \times\ sizeof(complex < double >) =\ 2^{33} \times 16 = 128\ Gb$$

Так как используется два вектора, то необходимая память будет составлять $256\ Gb$. Значит, максимально возможное количество кубитов составляет $n = 32$.

Результат выполнения.

Экспериментально было выявлено несоответствие теоретического значения максимального количества кубитов с фактическим. Фактически максимальное количество кубитов составило $n = 28$.

Результаты запусков программы приведены в таблице ниже.

Количество кубитов	*Номер кубита*	*Количество потоков*	*Время работы программы, с*	*Ускорение*
20	1	1	0,192960	1,000000
		2	0,105154	1,835023
		4	0,061802	3,122224
		8	0,051086	3,777160
		16	0,068897	2,800689
		32	0,042202	4,572303
		64	0,028644	6,736578
	2	1	0,192869	1,000000
		2	0,105157	1,834105
		4	0,061862	3,117755
		8	0,040696	4,739285
		16	0,040337	4,781492
		32	0,035047	5,503153

		64	0,028018	6,883766
	20	1	0,208298	1,000000
		2	0,113364	1,837426
		4	0,068649	3,034233
		8	0,050560	4,119810
		16	0,042474	4,904118
		32	0,033057	6,301263
		64	0,028944	7,196711
24	1	1	3,100720	1,000000
		2	1,687650	1,837300
		4	0,995247	3,115528
		8	0,647066	4,791969
		16	0,600099	5,167018
		32	0,497466	6,233029
		64	0,445437	6,961074
	2	1	3,097120	1,000000
		2	1,687860	1,834939
		4	0,983498	3,149086
		8	0,648128	4,778562
		16	0,581044	5,330270
		32	0,525419	5,894569
		64	0,447163	6,926154
	24	1	3,432300	1,000000
		2	1,852110	1,853184
		4	1,070410	3,206528
		8	0,685177	5,009363
		16	0,623205	5,507498
		32	0,505491	6,790039
		64	0,439672	7,806501
28	1	1	49,560900	1,000000
		2	27,176300	1,823681
		4	15,919200	3,113278
		8	10,613800	4,669477
		16	8,636140	5,738779
		32	7,605033	6,516856
		64	6,774822	7,315454
	2	1	49,435212	1,000000
		2	27,432443	1,802071
		4	15,732633	3,142208
		8	10,462344	4,725061
		16	8,242345	5,997712
		32	7,124234	6,939021
		64	6,523524	7,577992

	28	1	50,001243	1,000000
		2	27,171640	1,840200
		4	15,395726	3,247735
		8	10,824274	4,619362
		16	8,427484	5,933117
		32	7,814545	6,398484
		64	6,124797	8,163738

Исходный код

```
#include <iostream>
#include <vector>
#include <complex>
#include <cmath>
#include <cstdlib>
#include <string>
#include <omp.h>
#include <fstream>

using namespace std;

// количество кубитов
#define COUNT_QUBIT 20
// номер кубита , по которому проводится преобразование
#define NUM_QUBIT 1

// проверяет на 0 или 1 бит на numQubit-ном месте числа number
int checkBit(unsigned long long number, int numQubit) {
	unsigned long long mask = 1 << (COUNT_QUBIT - numQubit);
	if ((number & mask) == 0)
		return 0;
	return 1;
}

// возвращает число в 10-ой с.с. , которое получится после постановки 0 или 1 на numQubit-ный бит числа number
unsigned long long putZeroOrOne(int zeroOrOne, unsigned long long number, int numQubit) {
	unsigned long long mask = 1 << (COUNT_QUBIT - numQubit);
	if (zeroOrOne == 1)
		return (number | mask);
	mask = ~mask;
	return (number & mask);
}

// функция однокубитного преобразования
vector <complex <double> > oneQubitTransformation(vector <complex <double> > &vec, vector <vector <complex <double> > > &matrix, int numQubit) {
	vector <complex <double> > vecTransformed(vec.size());
	#pragma omp parallel for
	for (long long i = 0; i < vec.size(); i++)
		vecTransformed[i] = matrix[checkBit(i, numQubit)][0] * vec[putZeroOrOne(0, i, numQubit)] +
			matrix[checkBit(i, numQubit)][1] * vec[putZeroOrOne(1, i, numQubit)];
	return vecTransformed;
}

int	main(int argc, char **argv) {
	// инициализация вектора случайными значениями
```

```
46.       unsigned long long vectorSize = pow(2, COUNT_QUBIT);
47.       vector <complex <double> > vec(vectorSize);
48.       #pragma omp parallel for
49.       for (long long i = 0; i < vec.size(); i++)
50.             vec[i] = complex<double> (1. / rand(), 1. / rand());
51.       // инициализация матрицы преобразования Адамара
52.       vector <vector <complex <double> > > matrix(2);
53.       matrix[0].resize(2);
54.       matrix[1].resize(2);
55.       #pragma omp parallel for collapse(2)
56.       for (long long i = 0; i < 2; i++)
57.             for (long long j = 0; j < 2; j++) {
58.                   matrix[i][j] = 1. / sqrt(2);
59.                   if (i == 1 && j == 1)
60.                         matrix[i][j] = - matrix[i][j];
61.             }
62.       // вызов функции однокубитного преобразования
63.       double begin = omp_get_wtime();
64.       vector <complex <double> > vecTransformed = oneQubitTransformation(vec,
   matrix, NUM_QUBIT);
65.       double end = omp_get_wtime();
66.       // вывод времени работы программы в файл
67.       ofstream fout(argv[1], ios_base::app);
68.       fout << end - begin << endl;
69.       fout.close();
70.       // вывод нового вектора
71.       // for (long long i = 0; i < vecTransformed.size(); i++)
72.       //     cout << vecTransformed[i] << endl;
73.       return 0;
74. }
```