Параллельный алгоритм моделирования идеального квантового алгоритма Гровера

Корж Оксана, Чернявский Андрей, Андреев Дмитрий, Коробков Сергей

Московский государственный университет имени М.В.Ломоносова

Введение

- Квантовая информатика является относительно молодой и очень бурно развивающейся областью современной науки.
- Из важнейших направлений квантовой информатики можно выделить квантовые вычисления, квантовую криптографию и моделирование квантовых систем.
- При успешном создании квантового компьютера реализуемые на нем алгоритмы позволят решать некоторые важные задачи существенно быстрее, нежели на классических компьютерах.

О чем этот доклад?

- С точки зрения параллельных вычислений базовые алгоритмы квантовых вычислений относятся к классу DIC (Data Intensive Computing).
- Главным свойством задач этого класса является существенное преобладание чтения-записи данных по сравнению с количеством вычислений.
- При проведении каждой одно-, двух- и т.д. кубитных операции происходит изменение всего вектора состояний (фактически всей задействованной оперативной памяти). При этом доступ к данным осуществляется в произвольном порядке: порядок зависит от последовательности номеров кубитов, для которых выполняется преобразование.
- Моделирование квантовых вычислений – хорошая задача для анализа вычислительных возможностей суперкомпьютера
- Предлагается рассмотреть моделирование квантовых вычислений на суперкомпьютере с распределенным хранением данных. Использован метод активных сообщений

Квантовые вычисления

- Однокубитная операция

$$\{a_i\} = \{a_0, a_1, \ldots, a_{2^n-1}\} \qquad U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$$

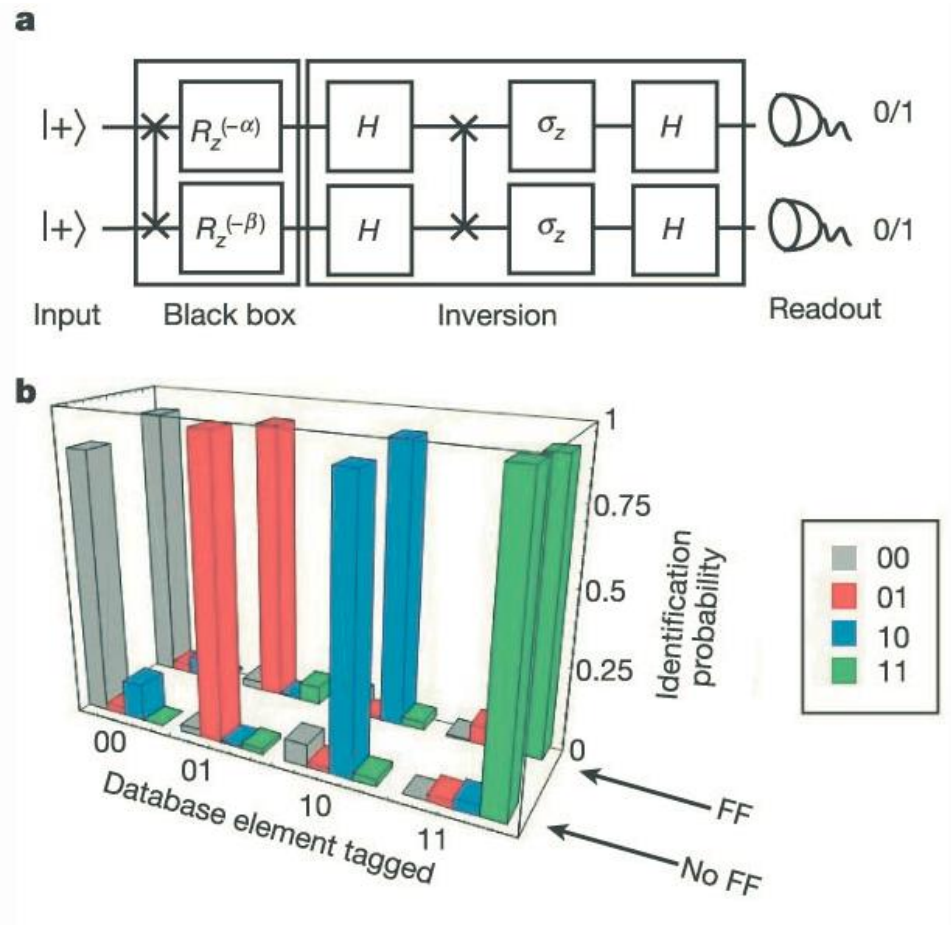
$$b_{i_1 i_2 \ldots i_k \ldots i_n} = \sum_{j_k=0}^{1} u_{i_k j_k} a_{i_1 i_2 \ldots j_k \ldots i_n} = u_{00} a_{i_1 i_2 \ldots 0_k \ldots i_n} + u_{01} a_{i_1 i_2 \ldots 1_k \ldots i_n}$$

$$\begin{aligned} b_0 &= b_{\mathbf{00}} = u_{00} a_{\mathbf{00}} + u_{01} a_{\mathbf{10}} \\ b_1 &= b_{\mathbf{01}} = u_{00} a_{\mathbf{01}} + u_{01} a_{\mathbf{11}} \\ b_2 &= b_{\mathbf{10}} = u_{10} a_{\mathbf{00}} + u_{11} a_{\mathbf{10}} \\ b_3 &= b_{\mathbf{11}} = u_{10} a_{\mathbf{01}} + u_{11} a_{\mathbf{11}} \end{aligned}$$

Алгоритм Гровера

- Квантовый алгоритм решения задачи перебора, то есть нахождения решения уравнения f(x)=1, где f есть булева функция от *n* переменных.
- Предполагается, что функция задана в виде чёрного ящика, или оракула, то есть в ходе решения мы можем только задавать оракулу вопрос типа: «чему равна f на данном x», и после получения ответа использовать его в дальнейших вычислениях.
- GSA находит какой-нибудь корень уравнения, используя $\frac{\pi}{4}\sqrt{N}$
- обращений к функции f, с использованием O(n) кубитов.

$$N = 2^n$$



Предыдущие работы

- *J. Robert Burger, “New approachs to quantum computer simulaton in a classical supercomputer”, CoRR, Vol. Quant-ph/0308158 (2003)*

- В статье представлен алгоритм для моделирования на компьютере с общей памятью. В работе используется специальные структуры данных для хранения разреженных матриц, при помощи которых представляется эволюция квантовых состояний в ходе вычислений. В результате такого подхода было достигнуто моделирование 32 кубитов на довольно небольших ресурсах: 16 гигабайт памяти и 64 процессора. Но данный метод применим только для систем с общей памятью, где нет необходимости в пересылках между различными вычислительными узлами.

Предыдущие работы

- *Goran Negovetic, Marek Perkowski, Martin Lukac, Andrzej Buller, “Evolving quantum circuits and an FPGA-based Quantum Computing Emulator”, International Worksho on Boolean Problems, 2002*

- В статье показана возможность использовать реконфигурируемые вычислители для моделирования квантовых алгоритмов. Но и в этом случае размерность моделируемой системы ограничена памятью вычислителя.

Предыдущие работы

- *Frank Tabakin, Bruno Juliá-Díaz, ”QCMPI: A parallel environment for quantum computing”, Computer Physics Communications, 2009.*

- Для использования больших кластеров с распределенной памятью требуется осуществлять обмены между вычислительными узлами. Для этого наиболее часто используется протокол MPI. Примером реализации программного обеспечения удовлетворяющего данным условиям является набор программ QCMPI, написанный на языке Fortran с использованием MPI. Эта программа имеет большой набор операторов: стандартный оператор Паули, преобразование Адамара, контролируемое отрицание, фазовый сдвиг, и квантовое преобразование Фурье.
- Программная система использует библиотеки линейной алгебры BLAS, LAPACK, SCALAPACK. В случае моделирования квантовых вычислений целесообразно использовать более частные алгоритмы для разреженных матриц.

Предыдущие работы

- *Guido Arnold, Thomas Lippert, Nikolas Pomplun, Marcus Richter, “Large Scale Simulation of Ideal Quantum Computers on SMP-Clusters”, PARCO , pp. 447-454, 2005*

- В статье описывается аналогичный подход к реализации модели квантового компьютера. В этой статье показано, что при моделировании 37 кубитного квантового компьютера можно добиться эффективности в 1 квантовую операцию за 10 секунд.

- *World record: German supercomputer simulates quantum computer http://phys.org/news189231849.html (дата обращения 01.12.2012г.)*

- В 2010 году было выполнено моделирование алгоритма Шора на суперкомпьютере Jugene в Суперкомпьютерном центре Юлиха. Пиковая производительность этого суперкомпьютера на тот момент составляла порядка одного петафлопса, объем оперативной памяти составлял порядка 140 терабайт. Удалось выполнить моделирование 42 кубитов.

Реализация параллельной версии алгоритма с помощью библиотеки DISLIB

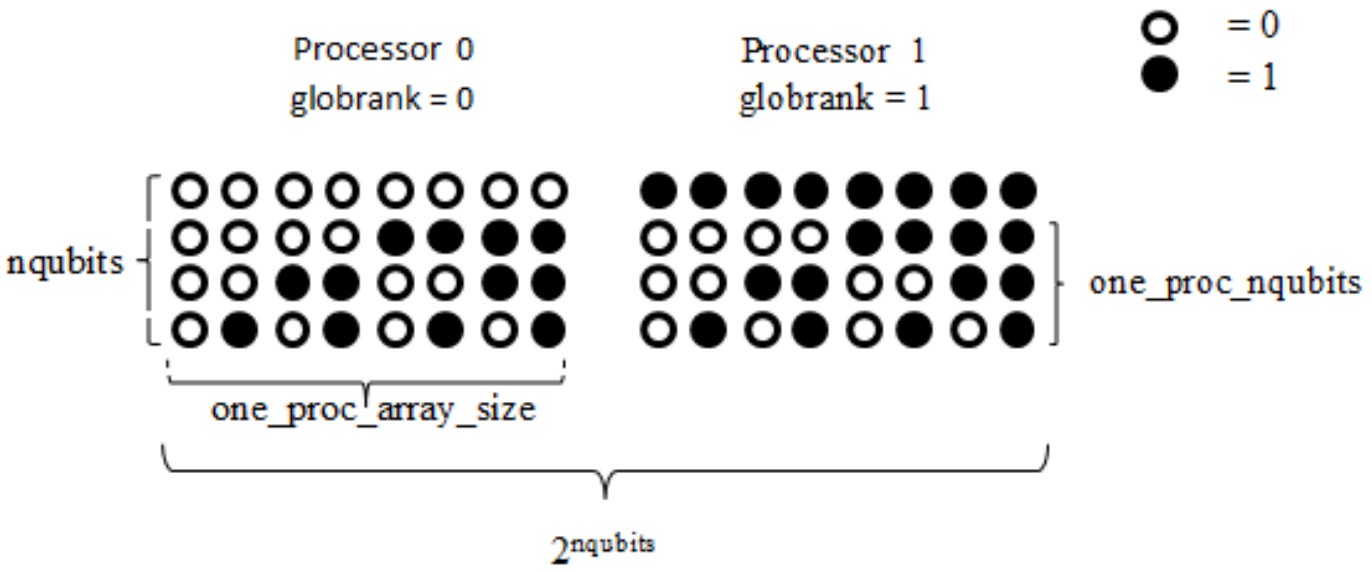
- Особенность квантовых вычислений с точки зрения реализации для параллельной вычислительной системы – необходимость постоянного нерегулярного доступа к данным, которые хранятся распределенно. Поэтому использование механизма односторонних активных сообщений представляется перспективным.

- Основной проблемой в реализации является размер хранимых данных для преставления текущего вектора состояний. Для выполнение одной операции одно-, двух-, трехкубитного преобразования требуется хранить два массива комплексных чисел двойной точности. Размер каждого массива – $2^{nqubits}$, где nqubits – количество моделируемых кубитов. В случае моделирования алгоритма Гровера размер массивов $2^{2*nqubits}$ элементов.

- Реализация на персональном компьютере – максимум 28 кубитов, реализация на суперкомпьютере Ломоносов – максимум 40-42 кубитов.

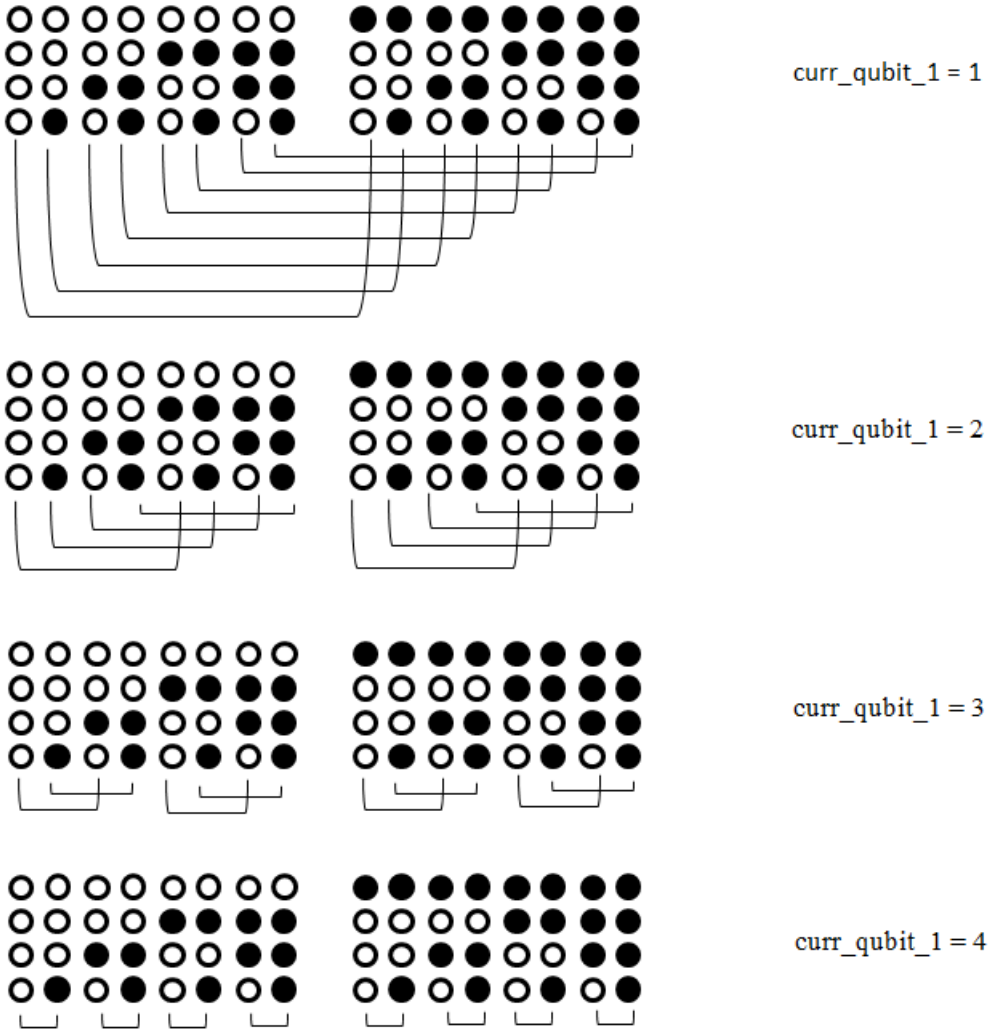
Реализация параллельной версии алгоритма с помощью библиотеки DISLIB

- Показан пример хранения вектора состояний. В примере показано хранение данных для случая двух процессоров и 4 кубитов.



Расположение данных по процессорам

- Показаны необходимые обращения к данным для различных номеров кубитов.
- Можно видеть, что для приведённого примера обмен данными между процессорами потребуется только в случае, если номер кубита, по которому проводится преобразование, равен одному.
- В случае использования модели активных сообщений такая структура данных определяет, какому процессору нужно послать значение текущего обрабатываемого элемента.



Реализация параллельной версии алгоритма с помощью библиотеки DISLIB

- Для каждого элемента массива нужно посчитать сумму двух слагаемых: свое текущее значение, умноженное на некоторый коэффициент и значение элемента с противоположным битом, также умноженное на некоторый коэффициент.
- Поскольку не определен порядок, в котором будет происходить суммирование, потребуется дополнительный массив, куда будут записаны результаты промежуточного суммирования.
- Если на процессоре не оказалось нужных данных для суммирования, то мы отправляем свой элемент на процессор, номер которого отличатся в бите, соответствующем номеру активного кубита. И соответственно ждем от этого процессора его элемент.
- При этом получение элемента может произойти раньше, чем отправка. Для этого необходимо выполнять барьерную синхронизацию после отправки и получения элемента.
- Для моделирования двух- и трехкубитных преобразований используется тот же принцип распараллеливания вычислений. Массив состояний равномерно разделяется по процессорам. Далее при изменении элементов определяется, какие являются локальными, а какие необходимо отправить / получить.

```
// one qubit evolution function, current modification qubit is curr_qubit_1
// current matrix for transformation is in curr_U1
void one_qubit_evolution(int curr_qubit_1)
{
    long long tmp_send_proc_ind; // receiving processor number
    long long tmp_U_ind; // local index with opposite bit
    elem_ind tmp_send; // data for sending
    // output array initialization
    for(long long i=0;i<one_proc_array_size;i++) {out[i] = 0;}
    shmem_barrier_all();
    // modification of all local array elements
    for(long long i=0;i<one_proc_array_size;i++)
    {
        // required element is non-local
        if ((nqubits - curr_qubit_1) >= one_proc_nqubits)
        {
            // data for sending
            tmp_send.elem = in[i]; tmp_send.ind = i;
            // receiving processor number estimation
            tmp_send_proc_ind =
            globrank ^ (1L << (nqubits - curr_qubit_1 - one_proc_nqubits));
            // if bit in curr_qubit_1 position is 0
            if (tmp_send_proc_ind > globrank)
            {
                // current element addition
                out[i] += curr_U1[0][0] * in[i];
                    shmem_send(&tmp_send, 1, sizeof(elem_ind), tmp_send_proc_ind);
            }
            else // if bit in curr_qubit_1 position is 1
            {
                // current element addition
                out[i] += curr_U1[1][1] * in[i];
                    shmem_send(&tmp_send, 1, sizeof(elem_ind), tmp_send_proc_ind);
            }
        }
        else //all elements are at the current processor
        {   //local index with opposit bit
            tmp_U_ind = i ^ (1L<< (nqubits - curr_qubit_1));
            //if bit in curr_qubit_1 position is 1
            if ( i > tmp_U_ind )
                out[i] += curr_U1[1][1] * in[i] + curr_U1[1][0] * in[tmp_U_ind];
            //if bit in curr_qubit_1 position is 0
            else out[i] += curr_U1[0][0] * in[i] + curr_U1[0][1] * in[tmp_U_ind];
        }
    }
    // all processes are finished
    shmem_barrier_all();
}
```

Оптимизация 1

- Часто получается так, что матрица коэффициентов разреженная, таким образом можно не пересылать элементы, которые заведомо будут умножаться в итоговой сумме на ноль.
- Также вектор исходных состояний часто равен {1,0,0,…,0}, поэтому так же можно экономить на посылке нулевых элементов.
- Такая оптимизация нарушает симметричность посылок. Однако это допустимо в модели активных сообщений. Если в процессе не выполнилась отправка, то получение активируется вызовом барьера.

Тестирование на системе Ломоносов

- На первом этапе тестирования реализованного алгоритма на суперкомпьютере Ломоносов требовалось провести оценку максимально возможного числа кубитов, которые получится моделировать.
- Это число ограничено размером данных, которые поместятся в оперативную память.
- Теоретическая оценка показывает, что в текущей конфигурации суперкомпьютера и при использовании разработанных алгоритмов максимально возможный размер моделирования составляет 40 кубитов.
- На 32 узлах возможно моделирование 33 кубитов. Максимально возможное число кубитов, доступных для моделирования на одном процессоре – 29 и 30 (зависит от типа процессора).

Тестирование на системе Ломоносов

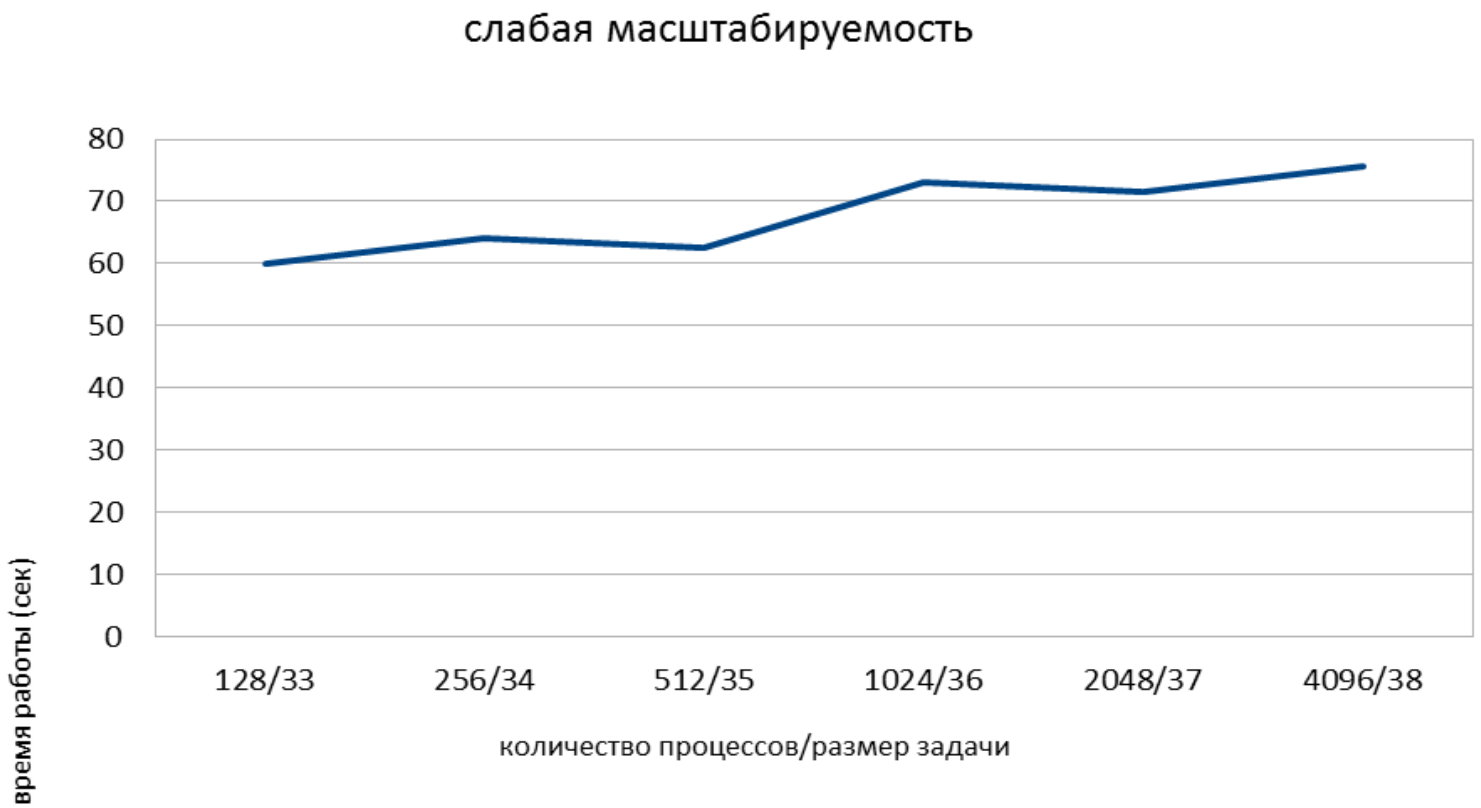
- Далее интерес представляла оценка среднего времени выполнения одного квантового преобразования.
- Тестирование проводилось для преобразования Адамара. Проводилось n-кубитное преобразование и время работы делилось на количество кубитов.
- В таблице приведено время работы программы в секундах для моделирования от 24 до 33 кубитов. Значение SF означает, что задача данного размера не может быть смоделирована на указанном количестве процессоров из-за недостаточного объема памяти.
- Тестирование на партиции test (каждый В/У – 12ГБ памяти)

	24	25	26	27	28	29	30	31	32	33
4	0.0319	0.0635	0.1275	0.2522	0.5032	1.1414	2.2330	SF	SF	SF
8	0.0102	0.0318	0.6473	0.1261	0.2518	0.5105	1.1423	2.2604	SF	SF
16	0.0083	0.0171	0.3213	0.0652	0.1293	0.2517	0.0503	1.1443	2.2896	SF
32	0.0043	0.0085	0.0160	0.0335	0.0628	0.1267	0.2504	0.5047	1.0718	2.3216

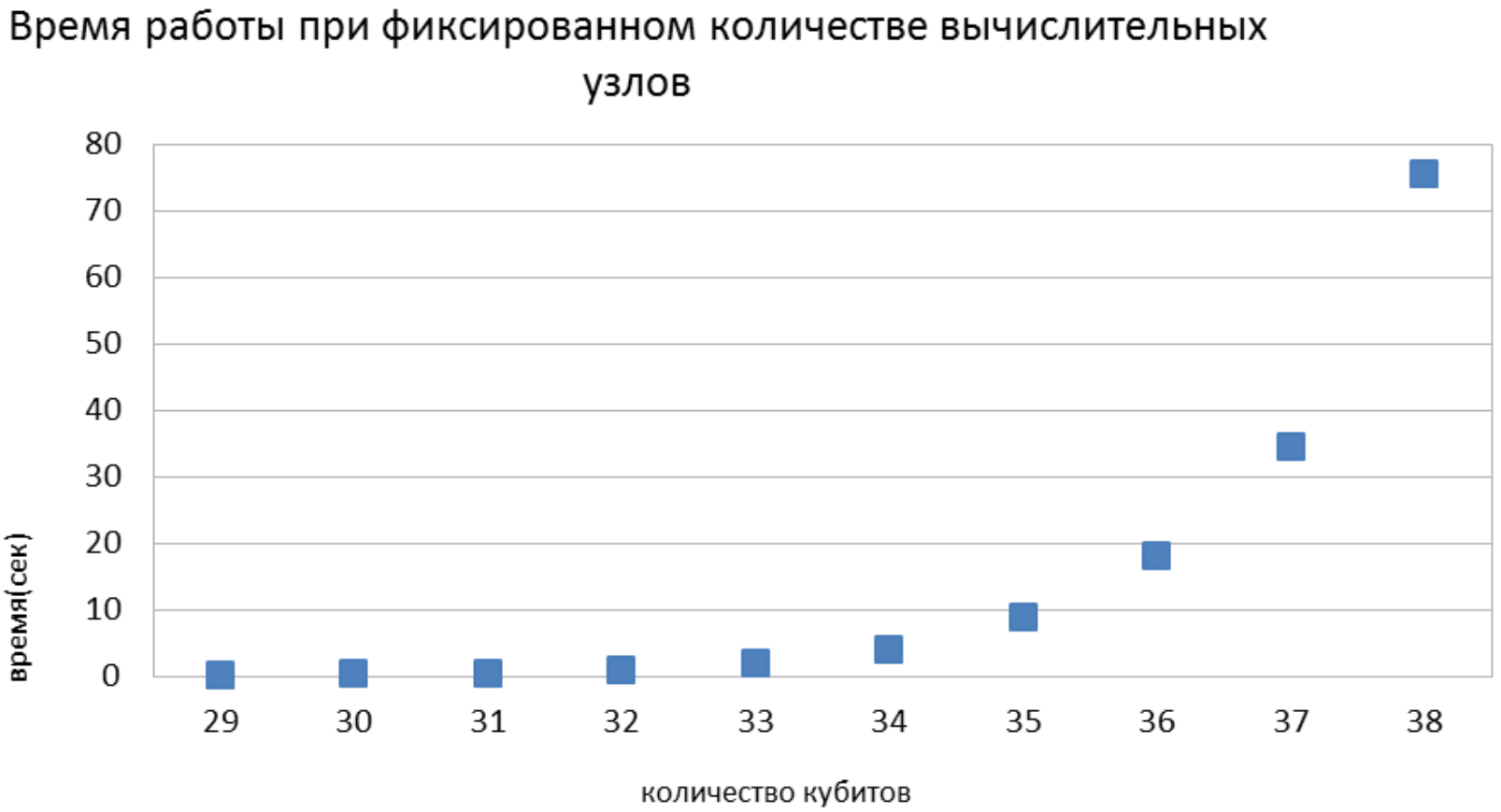
Оценка времени работы однокубитного преобразования

- Напомним, что в Исследовательском центре Юлиха при моделировании 37 кубитного квантового компьютера была получена эффективность в 1 квантовую операцию за **10 секунд**.

Слабая масштабируемость



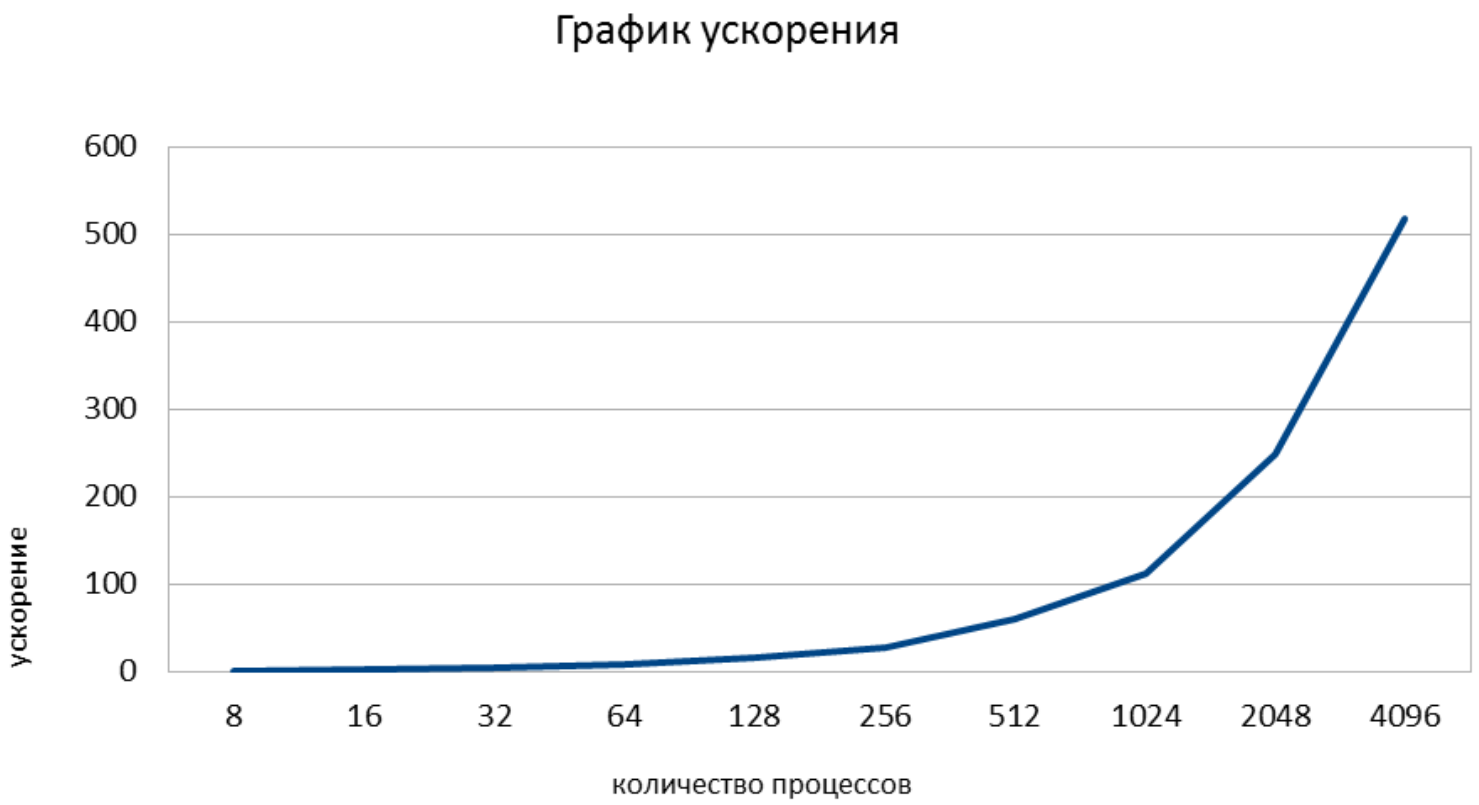
(512 узлов, 4096 процессов)



Время работы квантового алгоритма Гровера

Grover	14	15	16
1	10971.4132	SF	SF
2	5582.2765	SF	SF
4	2821.2334	17628.4555	SF
8	1452.7525	8876.5264	SF
16	746.7995	4499.1933	13932.3842
32	383.9903	2308.4533	6737.3245
64	198.3245	1274.4882	3367.0641

Время работы квантового алгоритма Гровера



- Для оценки сильной масштабируемости размер задачи фиксирован (максимально возможная задача, для которой хватит пямяти одного вычислительного узла) – 14 кубитов (размер данных $O(2^{2n})$).

Время работы квантового алгоритма Гровера

Заключение и дальнейшие планы

Представлено моделирование некоторых базовых алгоритмов квантовых вычислений на суперкомпьютере. Для моделирования было разработано программное обеспечение, использующее механизм передачи активных сообщений для межпроцессорных коммуникаций. Использована библиотека DISLIB. Проведено моделирование однокубитной квантовой операции, квантового преобразования Фурье и квантового алгоритма Гровера. Во всех экспериментах показана хорошая масштабируемость разработанной параллельной программы.

Планы:

- Реализация «научных» вычислительных задач (до этого задачи были модельными). В процессе задача: Расчет редуцированной матрицы плотности для чистых состояний
- Применение алгоритмов сжатия для хранения данных
- Когда-нибудь возможно создание фреймворка и графического интерфейса

Спасибо за внимание

oxana@cs.msu.su

sqi.cs.msu.ru