УДК 004.272.2

МОДЕЛИРОВАНИЕ РАБОТЫ ИДЕАЛЬНОГО КВАНТОВОГО КОМПЬЮТЕРА НА СУПЕРКОМПЬЮТЕРЕ “ЛОМОНОСОВ”

О. В. Корж[1], Д. Ю. Андреев[2], А. А. Корж[3], С. В. Коробков[1], А. Ю. Чернявский[4]

Одной из задач, решение которых предполагается получать с помощью экзафлопсного суперкомпьютера, является построение компьютера на новых принципах для достижения существенного прогресса в скорости вычислений. В настоящей статье представлено моделирование работы идеального квантового компьютера на суперкомпьютере “Ломоносов”. Предложен эффективный алгоритм распараллеливания вычислений при одно-, дву- и трехкубитных преобразованиях с использованием библиотеки DISLIB. В качестве примера моделирования рассматривается квантовый алгоритм Гровера и квантовое преобразование Фурье. Работа выполнена при финансовой поддержке РФФИ (гранты 12–07–31229 и 12–01–31274). Статья рекомендована к публикации Программным комитетом Международной научной конференции “Параллельные вычислительные технологии” (ПаВТ–2013; http://agora.guru.ru/pavt2013).

Ключевые слова: суперкомпьютер, квантовая информатика, алгоритм Гровера, квантовое преобразование Фурье, параллельные алгоритмы.

1. Введение. При успешном создании квантового компьютера реализуемые на нем алгоритмы позволят решать некоторые сложные вычислительные задачи существенно быстрее, нежели на классических компьютерах. Так, алгоритм Шора [1] позволяет раскладывать числа на простые множители за полиномиальное время, а алгоритм Гровера [2] позволяет решать любые переборные задачи за время, примерно равное квадратному корню из времени, затрачиваемого на классических компьютерах. Для создания полномасштабного квантового компьютера, несомненно, необходимо моделирование его работы. Особенно важно моделирование на основе реальных физических систем с учетом квантового шума, который является основным препятствием на пути реализации квантовых битов (кубитов) и вентилей. Отметим, что с ростом числа кубитов необходимые вычислительные ресурсы растут экспоненциально. В связи с этим фактом для решения многих задач даже с не очень большим числом кубитов невозможно обойтись без использования суперкомпьютеров.

Данная работа посвящена моделированию идеальных квантовых схем и является очередным шагом к решению важных практических задач квантовой информатики. В настоящей статье рассматривается моделирование алгоритма Гровера, а также важнейшей подпрограммы квантовых алгоритмов — квантового преобразования Фурье, используемого в алгоритме Шора и других алгоритмах, основанных на выделении периода функций [3, 4].

С точки зрения параллельных вычислений базовые алгоритмы квантовых компьютеров 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-, дву- и т.д. кубитной операции происходит изменение всего вектора состояния. При этом доступ к данным осуществляется в произвольном порядке: порядок зависит от последовательности номеров кубитов, для которых выполняется преобразование. Квантовые вычисления могут быть эффективно смоделированы на машинах с общей памятью, однако очевидно, что размеры современных машин с общей памятью не позволят выполнить моделирование существенного количества кубитов. В данной статье предлагается рассмотреть моделирование квантовых вычислений на суперкомпьютере с распределенным хранением данных. Для реализации параллельной версии программы предлагается использовать метод активных

[1] Московский государственный университет им. М. В. Ломоносова, факультет вычислительной математики и кибернетики, Ленинские горы, д. 1, 119991, Москва; О. В. Корж, ассистент, e-mail: oxanad@mail.ru; С. В. Коробков, программист, e-mail: korobkovserg@gmail.com

[2] Вычислительный центр им. А. А. Дородницына РАН, ул. Вавилова, д. 40, 119333, Москва; аспирант, e-mail: andreevd@cs.msu.ru

[3] ОАО “Т-платформы”, Ленинский просп., д. 113/1, 117198, Москва; руководитель группы, e-mail: anton@korzh.ru

[4] Физико-технологический институт РАН, Нахимовский просп., д. 36/1, 117218, Москва; науч. сотр., e-mail: andrey.chernyavskiy@gmail.com

сообщений и, в частности, библиотеку DISLIB [5]. Преимуществом такого подхода является возможность оптимизации пересылок данных между процессорами за счет использования низкоуровневых протоколов передачи данных в коммуникационных сетях. Высокоуровневый интерфейс библиотеки позволяет скрыть от пользователя особенности протоколов нижнего уровня, используемых на той или иной суперкомпьютерной архитектуре. Использование стандартного протокола MPI для такого рода задач представляется нецелесообразным, поскольку этот протокол является неэффективным для коммуникационно-сложных задач [6].

Статья включает в себя краткий обзор существующих подходов к моделированию квантовых вычислений на суперкомпьютерах, краткое описание квантовых алгоритмов, для которых проводится моделирование (алгоритма Гровера и квантового преобразования Фурье), и описание программной реализации с использованием протокола активных сообщений. В разделе 5 обсуждаются результаты экспериментов на суперкомпьютере "Ломоносов", а также приводится анализ масштабируемости разработанных методов.

2. Обзор существующих подходов. При моделировании квантовых вычислений на суперкомпьютере количество необходимой памяти растет пропорционально экспоненте от числа кубитов. Соответственно, возможности моделирования квантового компьютера на классических устройствах весьма ограничены. Так, на современных персональных компьютерах доступное число кубитов составляет порядка 25–28, а для моделирования 35 идеальных кубитов уже требуется более 1 терабайта оперативной памяти.

Использование суперкомпьютера для моделирования квантовых вычислений требует специализированного программного обеспечения, которое позволит легко эмулировать выбранную квантовую схему. Естественно, для эффективного использования такое программное обеспечение должно быть параллельным. В статье [7] представлен алгоритм моделирования квантовых вычислений на компьютере с общей памятью. В этой работе используются 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. Однако указанный алгоритм применим только для систем с общей памятью, когда нет необходимости в пересылках между различными вычислительными узлами.

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 таким условиям, является написанный на языке Фортран с использованием MPI набор программ QCMPI [8]. Этот набор предоставляет большой выбор операторов: стандартный оператор Паули, преобразование Адамара, контролируемое отрицание, фазовый сдвиг, квантовое преобразование Фурье и др. Используются библиотеки линейной алгебры BLAS [9], LAPACK [10] и SCALAPACK [11]. В случае моделирования квантовых вычислений целесообразно использовать специальные алгоритмы для разреженных матриц.

В статье [12] 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. Кроме того, возможно использование реконфигурируемых вычислителей [13] для моделирования квантовых алгоритмов, однако и в этом случае размерность моделируемой системы ограничена памятью вычислителя.

В 2010 г. было выполнено моделирование алгоритма Шора на суперкомпьютере Jugene в Суперкомпьютерном центре Юлиха [14]. Пиковая производительность этого суперкомпьютера на тот момент составляла порядка одного петафлопса, объем оперативной памяти составлял порядка 140 терабайт. Удалось выполнить моделирование 42 кубитов. В случае же реализации подхода с хранением разреженных матриц и работой с такими матрицами с использованием специализированных протоколов обмена данными можно добиться увеличения размерности моделируемого квантового компьютера.

3. Квантовый алгоритм Гровера и квантовое преобразование Фурье. Задачей данной работы является суперкомпьютерное моделирование многокубитных квантовых схем для алгоритма Гровера и квантового преобразования Фурье. Кратко опишем формализм квантовых вычислений и рассматриваемые схемы.

3.1. Общая схема квантовых вычислений. Подробное описание формализма квантовых вычислений можно найти, например, в книгах [3, 4].

Состояние одного кубита квантового компьютера определяется нормированным на единицу вектором двумерного комплексного пространства $\mathbb{C}^2$: $a\big|0\big\rangle + b\big|1\big\rangle$, $|a|^2 + |b|^2 = 1$, где через векторы $\big|0\big\rangle$ и $\big|1\big\rangle$ обозначаются базисные векторы рассматриваемого пространства.

Состояние n-кубитной квантовой системы определяется тензорным произведением однокубитных

пространств $\left(\mathbb{C}^2\right)^{\otimes n} = \underbrace{\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \ldots \otimes \mathbb{C}^2}_{n}$, соответствующие векторы имеют вид

$$\sum_{i_1,i_2,\ldots,i_n=0}^{1} c_{i_1,i_2,\ldots,i_n} \left|i_1 i_2 \ldots i_n\right\rangle. \tag{1}$$

В каждый момент времени состояние меняется под действием некоторого унитарного преобразования. Обычно преобразования действуют лишь на малое число кубитов (в настоящей работе рассматриваются одно- и двукубитные преобразования). С формальной точки зрения такие преобразования имеют вид $U \otimes I$, где U — матрица, действующая на выбранные кубиты или кубит, и I — единичная матрица, действующая на остальные кубиты. Квантовый алгоритм представляет собой последовательное применение таких преобразований. Входные данные подаются либо в виде начального состояния (вектора) системы, либо в виде выбора применяемых преобразований. После окончания работы алгоритма производится так называемое измерение, дающее для состояний вида (1) n-битный ответ $\left\{i_1 i_2 \ldots i_n\right\}$ с вероятностью $\left|c_{i_1,i_2,\ldots,i_n}\right|$. Таким образом, квантовый алгоритм должен увеличивать коэффициент $c_{i_1,i_2,\ldots,i_n}$ при правильном ответе задачи.

Последовательность квантовых операций, которую и представляет собой алгоритм, удобно представлять графически в виде так называемых квантовых схем, пример одной из которых приведен на рис. 1.

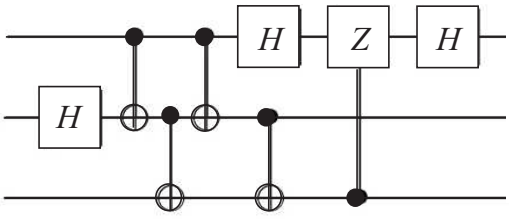

Рис. 1. Пример квантовой схемы. Каждая линия соответствует одному кубиту

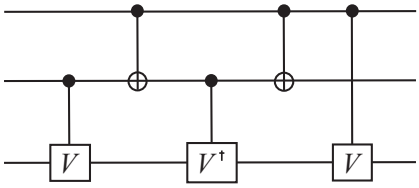

Рис. 2. Реализация элемента Тоффоли

3.2. Алгоритм Гровера. Пусть задана булева функция $f(x)$, $x \in \{0,1\}^n$. Необходимо найти такое значение x, что $f(x) = 1$. Очевидно, что классический алгоритм в среднем решает задачу за 2^{n-1} обращений к функции f (оракулу), причем улучшить порядок $O\left(2^n\right)$ невозможно. Алгоритм Гровера позволяет решить задачу за время $O\left(2^{n/2}\right)$. Важно отметить, что для работы алгоритма Гровера необходим так называемый квантовый оракул, переводящий базисное квантовое состояние $\left|x\right\rangle$ в $(-1)^{f(x)}\left|x\right\rangle$. Несложно показать, что такой оракул может быть легко создан в виде квантовой схемы для любой булевой функции, заданной в виде схемы из функциональных элементов. В силу ограниченности статьи мы не будем приводить квантовую схему алгоритма Гровера и объяснение действия его работы — соответствующую информацию можно найти в [3, 4]. Приведем лишь особенности реализации алгоритма.

Важной подпрограммой (или подсхемой) алгоритма Гровера является инверсия относительно нулевого состояния $2\left|0^n\right\rangle\left\langle 0^n\right| - I$. В нашей работе данное преобразование реализуется на основе методов раздела 4.3 книги [4] при помощи преобразований Тоффоли и контролируемого преобразования $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$. Само преобразование Тоффоли реализуется в виде схемы, указанной на рис. 2, где $V = \begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$ обозначает корень из операции NOT.

Оракул в представленных тестовых задачах реализуется в виде обращения знака элемента вектора, соответствующего правильному ответу, однако при необходимости может быть заменен на произвольную квантовую схему, соответствующую классической схеме вычисления некоторой булевой функции. Кроме того, в большинстве тестов инверсия производится аналогично оракулу (обращением знака при соответствующих амплитудах).

3.3. Квантовое преобразование Фурье. Квантовое преобразование Фурье является квантовым аналогом дискретного преобразования Фурье. Это преобразование играет ключевую роль в алгоритме Шора, который позволяет разложить число N на простые множители за время $O\left(\log^2 N \, \log^3(\log N)\right)$, что дает экспоненциальный рост скорости относительно наилучшего известного на данный момент классического алгоритма. Кроме того, на основе квантового преобразования Фурье строятся алгоритмы моделирования квантовых систем на квантовом компьютере [7].

Само преобразование имеет вид $|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{jk} |k\rangle$, $\omega = \frac{2\pi i}{2^n}$. Для соответствия с видом (1) следует отметить, что под j и k подразумеваются двоичные записи этих чисел. Такое преобразование задается рекурсивной схемой, приведенной на рис. 3.

4. Реализация параллельной версии алгоритма с помощью библиотеки DISLIB. Особенность квантовых алгоритмов 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.

При разработке библиотеки DISLIB за основу была взята модель программирования SHMEM (SHared MEMory). Основными характеристиками модели программирования SHMEM являются: стиль программирования SPMD (одна программа и множество данных), использование модели односторонних коммуникаций (PUT и GET), использование глобальных барьеров для разделения фаз коммуникаций и вычислений. В таком виде эта модель оказалась крайне полезна и удобна для множества задач, решаемых ключевыми клиентами компаний Cray и SGI. Свое признание эта модель также получила с принятием стандарта MPI-2.2, в который были включены односторонние операции MPI RMA one-sided. Однако по разным причинам (большая популярность MPI-1, утомленность пользователей, плохая поддержка производителями суперкомпьютеров и т.п.) большого распространения эта модель не получила. Библиотека DISLIB расширяет модель программирования SHMEM, при этом показано, что на ряде задач такое расширение, с одной стороны, повышает продуктивность программирования, что в ряде случаев позволяет решить те задачи, которые в рамках прежней модели решены быть не могут, а с другой стороны, производительность и масштабируемость написанных программ находится на высоком уровне. Сказанное отличает предложенное расширение DISLIB от ряда программных моделей, таких как, например, язык UPC, который позволяет резко повысить продуктивность программирования, но не позволяет при этом добиться высоких уровней масштабирования, оставаясь, таким образом, больше средством для быстрого прототипирования, нежели практического использования для суперкомпьютерных приложений.

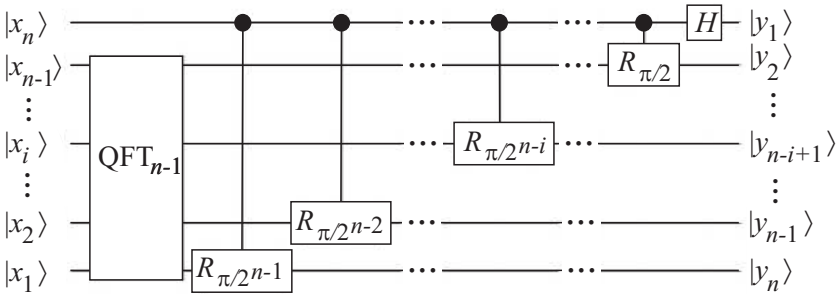

Рис. 3. Схема квантового преобразования Фурье

Ключевыми особенностями библиотеки DISLIB являются:

1) наличие расширенных операций PUT (односторонние активные сообщения);

2) наличие расширенных операций GET (двусторонние активные сообщения);

3) эффективная и прозрачная реализация агрегации сообщений;

4) многоступенчатая реализация передачи сообщений в многоядерных системах.

В линейной модели коммуникационной сети (linear performance model) стоимость посылки сообщения оценивается как $a + b{*}\mathrm{SZ}$, где SZ — это размер сообщения в байтах. Обычно параметр a оценивает задержку, а b — пропускную способность. В современных сетях, таких как Infiniband QDR/FDR, параметр a находится в диапазоне 3–4 мкс, а параметр b — менее одной наносекунды. Таким образом, для небольших сообщений (десятки байт) определяющим оказывается именно задержка a. В реальных сетях присутствует понятие конвейеризации сообщений, но даже в этом случае присутствует неконвейеризуемая часть задержки порядка нескольких микросекунд, например выдача команды через шину PCI-Express. Таким образом, замена посылки большого числа сообщений посылкой одного сообщения большого размера дает значительную экономию времени интерконнекта и процессора. Однако для многих современных и популярных моделей программирования, например MPI-1, реализация данной идеи требует изменения пользовательского интерфейса, иначе производительность может быть катастрофически низкой. Причиной этому является тот факт, что библиотека не может знать, какие сообщения и когда можно задерживать, а какие нет, а явного указания этого в модели программирования MPI не присутствует. В библиотеке ARMCI (Aggregate Remote Memory Copy Interface) пользователю предлагается явно указывать, какие сообщения агрегировать и в какой момент посылать, что негативно сказывается на продуктивности программирования. В модели программирования SHMEM глобальный барьер, помимо функции синхронизации процессов между собой (как MPI_Barrier), выполняет функцию разделения фаз коммуникаций и вычислений, необходимую для односторонних обменов, так как односторонние обмены не включают в себя

элемент синхронизации. Именно эта особенность и позволила совместить как удобство программирования, так и высокую масштабируемость и производительность в библиотеке DISLIB.

Основной проблемой в реализации является размер данных, хранимых для представления текущего вектора состояний. Для выполнения одной операции одно-, дву- и трехкубитного преобразования требуется хранить два массива комплексных чисел двойной точности. Размер каждого массива равен 2^{nqubits} элементов, где nqubits — количество моделируемых кубитов.

В случае моделирования алгоритма Гровера размер массивов равен $2^{2*\text{nqubits}}$ элементов (nqubits логических кубитов и nqubits вспомогательных). На рис. 4 показан пример хранения вектора состояний. В примере показано хранение данных для случая двух процессоров и 4 кубитов. Битовая длина номера элемента в массиве равна количеству используемых кубитов, при этом номер элемента можно представить как номер процессора, на котором находятся данные, и номер элемента в локальном массиве на каждом процессоре. В рассмотренном выше примере номер процессора соответствует первому биту. Таким образом, можно определять номер элемента в глобальном массиве по номеру элемента в локальном массиве и номеру процессора. Однако это накладывает то ограничение, что количество используемых процессоров должно быть степенью двух.

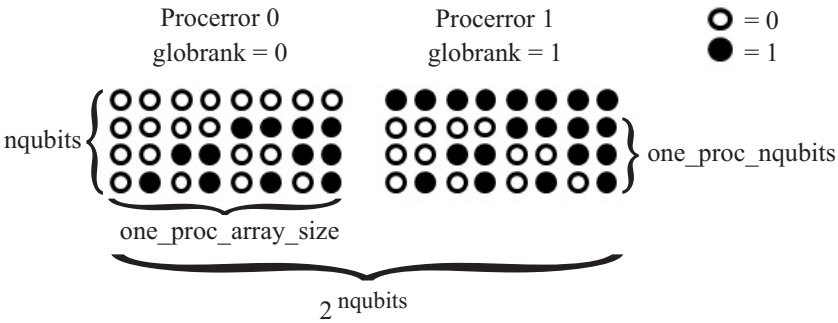

Рис. 4. Расположение данных по процессорам

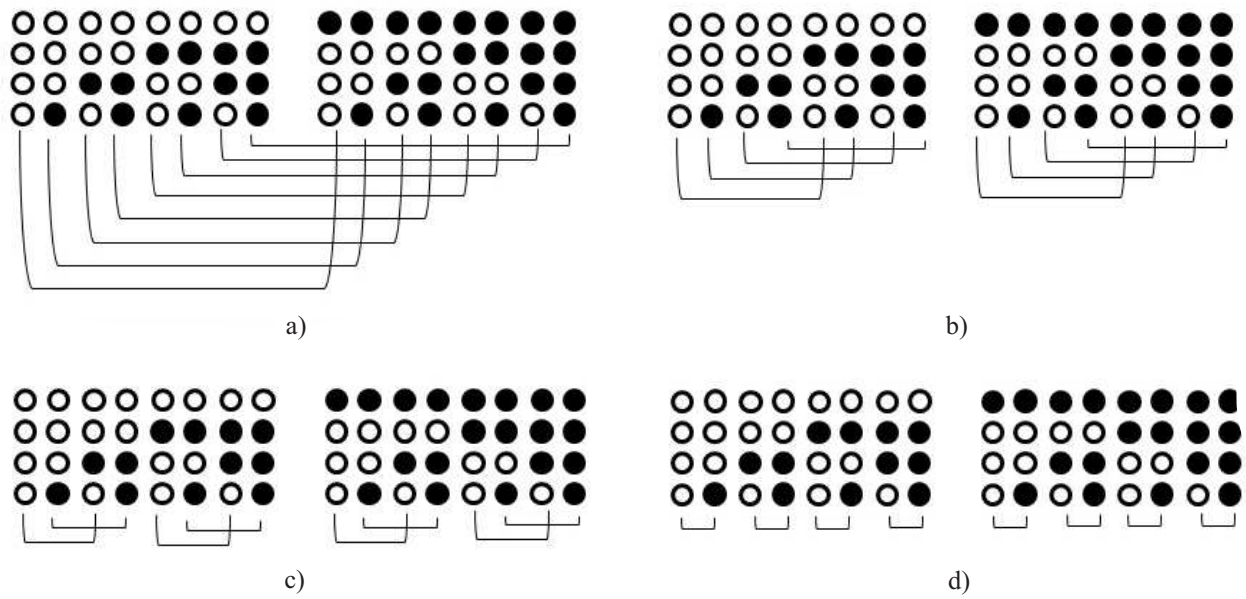

Рис. 5. Обмен данными при различных номерах активного кубита:
a) curr_qubit_1=1; b) curr_qubit_1=2; c) curr_qubit_1=3; d) curr_qubit_1=4

При реализации однокубитной операции происходит изменение всех элементов массива состояния, при этом для расчета нового значения элемента используется текущее значение и значение элемента, индекс которого в глобальном массиве состояний отличается на один бит. Причем отличающийся бит находится в позиции, которая соответствует номеру кубита, по которому осуществляется преобразование. Для части номеров кубитов нужный элемент находится на текущем процессоре, а часть элементов необходимо передать по сети с других процессоров. На рис. 5 показаны необходимые обращения к данным для различных номеров кубитов. Можно видеть, что для приведенного примера обмен данными между процессорами потребуется только в случае, когда номер кубита, по которому проводится преобразование, равен единице. В случае использования модели активных сообщений такая структура данных определяет, какому процессору нужно послать значение текущего обрабатываемого элемента. Соответственно, идея распа-

раллеливания однокубитной операции заключается в следующем. 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 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 всех элементов одной итерации.

Ниже приведен код функции, реализующей отправку сообщений.

Листинг 1. *Параллельная функция для однокубитного преобразования.*

```
// 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();
```

```
}
```

На следующем листинге показана функция-обработчик для этой функции отправки.

Листинг 2. *Хэндлер для функции однокубитного преобразования.*

```
typedef struct elem_ind {
        complexd elem;
        long long ind;
}e_i;

// handler for one_qubit_evolution function
void one_qubit_evolution_hhnl(int from,void* data,int sz)
{
        // received data
        elem_ind * temp = (elem_ind *)data;

        // if current modification bit is 1
        if ((globrank - from) > 0 )
                out[temp->ind] += curr_U1[1][0] * temp->elem;
        // if current modification bit is 0
        else out[temp->ind] += curr_U1[0][1] * temp->elem;
}
```

Часто оказывается, что матрица коэффициентов разрежена; таким образом, можно не пересылать элементы, которые заведомо будут умножаться в итоговой сумме на ноль. Кроме того, вектор исходных состояний часто равен $\{1, 0, 0, \ldots, 0\}$, поэтому так же можно экономить на посылке нулевых элементов. Такая оптимизация нарушает симметричность посылок, однако это допустимо в модели активных сообщений и не влияет на производительность. Если в процессе не выполнилась отправка, то получение активируется вызовом барьера.

Для моделирования дву- 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. Применяются аналогичные оптимизации с пересылками нулевых элементов и элементов, которые должны быть умножены на нулевые коэффициенты. С помощью определенных выше простейших функций реализуется набор операторов, таких как преобразование Адамара, фазовый сдвиг, CNOT и др.

В следующем листинге приведена реализация алгоритма Гровера, последовательно вызывающая приведенные выше операторы. Особенность реализации алгоритма — массив состояний имеет размер $2^{2*\mathrm{nqubits}}$ элементов. Это вдвое уменьшает количество доступных для моделирования логических кубитов. Аналогичным образом реализовано квантовое преобразование Фурье.

Листинг 3. *Реализация квантового алгоритма Гровера.*

```
const double pi = asin(1.0)*2;
//Number of algorithm steps
double r = pi*(sqrt((double)(1L<<(nqubits_2))))/4;

// Grover transformation
void Grover()
{
        Hn(); // n-qubits Hadamar transformation

        for (long long i=1; i< r; i++)
        {
                Oracle(); // Oracle function
                Hn(); // n-qubits Hadamar transformation
                PhaseInverse(); // phase inverse transformation
                Hn(); // n-qubits Hadamar transformation
        }
}
```

Реализованы идеальные квантовые алгоритмы. В дальнейшем планируется реализация возможности добавления шумов. Пользователю предоставляется библиотека, реализующая базовые квантовые преобра-

зования. Далее пользователь имеет возможность конструировать различные алгоритмы из этих преобразований. Библиотека легко расширяема. Переносимость библиотеки определяется возможностью установки библиотеки DISLIB на кластере. В дальнейшем предполагается разработка графического интерфейса для задания квантовой схемы с помощью графических нотаций.

5. Результаты тестирования алгоритмов на суперкомпьютере "Ломоносов". На первом этапе тестирования реализованного алгоритма на суперкомпьютере "Ломоносов" требовалось оценить максимально возможное число кубитов, которые удается моделировать. Это число ограничено размером данных, которые поместятся в оперативную память процессора. Теоретическая оценка показывает, что в текущей конфигурации суперкомпьютера и при использовании разработанных алгоритмов максимально возможный размер моделирования — 40 кубитов. На 32 узлах возможно моделирование 33 кубитов. Максимально возможное число кубитов, доступных для моделирования на одном процессоре — 29. Предварительные эксперименты по анализу характеристик разработанного алгоритма проводились на 32 узлах партиции test. В разделе 5.1 приведены результаты этого моделирования. Далее проводилось полномасштабное моделирование на партиции gpu. Результаты этих экспериментов приведены в разделе 5.2.

Таблица 1

Время работы одного квантового преобразования

	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

5.1. Тестовые эксперименты. Интерес представляет оценка среднего времени выполнения одного квантового преобразования. Тестирование проводилось для преобразования Адамара. Выполнялось n-кубитное преобразование, и время работы делилось на количество кубитов. В табл. 1 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.

На рис. 6 показан график ускорения, полученного в этом эксперименте. Полученное ускорение практически линейное для всех рассмотренных размеров задачи. Таким образом, можно говорить о хорошей масштабируемости этой задачи как по отношению к ее размеру, так и по отношению к увеличению количества процессоров.

Далее был рассмотрен эксперимент по моделированию квантового преобразования Фурье. Результаты работы программы приведены в табл. 2, в которой указывается значение времени (в секундах) в зависимости от количества процессоров и количества моделируемых кубитов.

Таблица 2

Время работы квантового преобразования Фурье (QFT)

QFT	28	29	30	31	32	33
8	17.4491	38.0417	90.2237	186.8465	SF	SF
16	8.5838	18.0346	40.3449	82.3250	192.9130	SF
32	4.3957	9.3806	19.5471	42.4178	97.0880	203.6334

При моделировании квантового преобразования Фурье так же была получена существенная масштабируемость приложения благодаря использованию механизма активных сообщений, при этом время моделирования пропорционально зависит от размера задачи.

В табл. 3 показано время работы программы по моделированию квантового алгоритма Гровера. Эта задача существенно более ресурсоемкая как по времени работы, так и по количеству требуемой памяти. В эксперименте так же наблюдается хорошая масштабируемость задачи (рис. 7).

5.2. Результаты моделирования в монопольном режиме доступа к суперкомпьютеру. Авторами проводилось моделирование больших размеров задачи на большем числе вычислительных узлов. Максимально моделируемое число кубитов составило 39. Для этого использовалось 2048 вычислительных узлов суперкомпьютера "Ломоносов". Приведем подробные результаты серии экспериментов с использо-

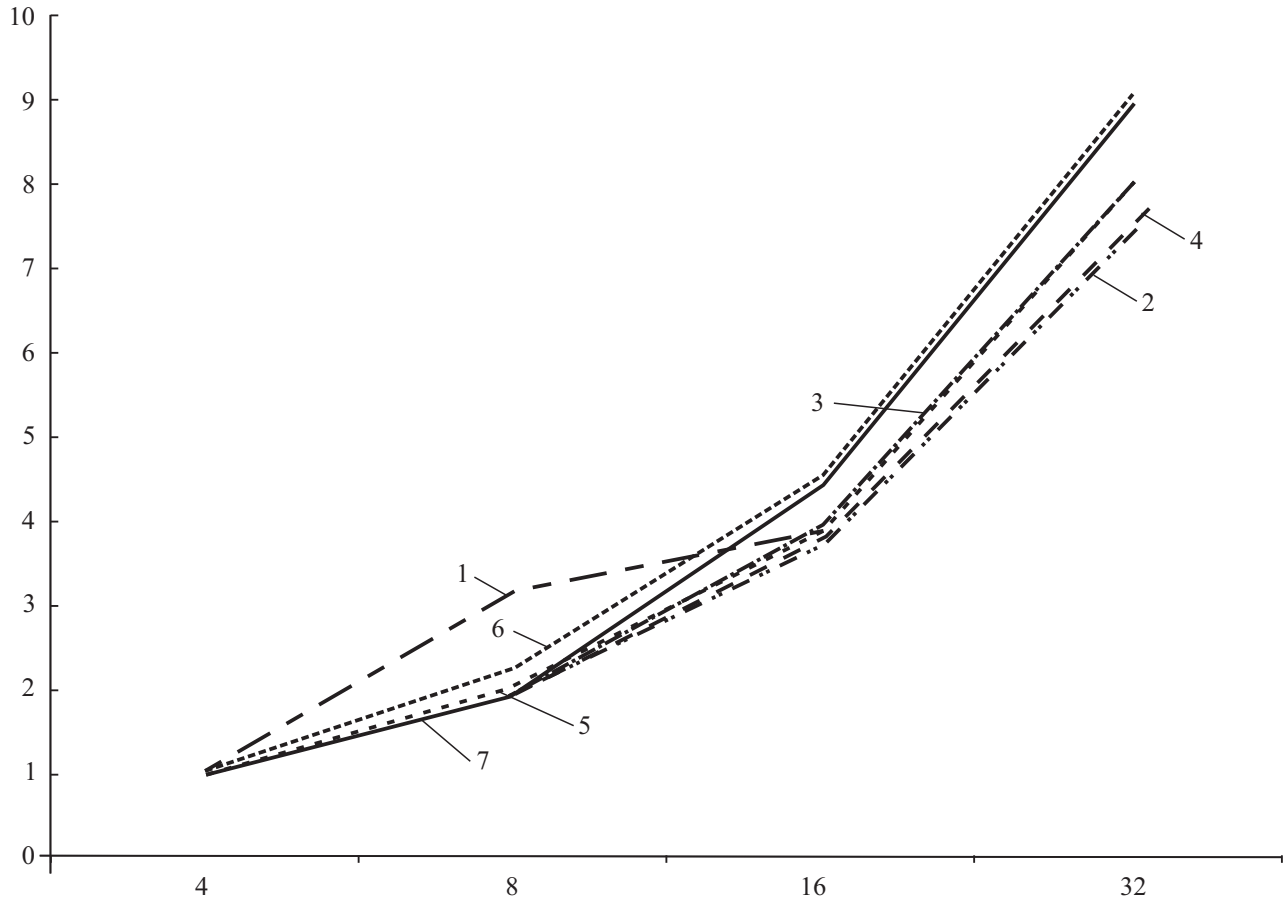

Рис. 6. График ускорения для однокубитного преобразования: 1) 24, 2) 25, 3) 26, 4) 27, 5) 28, 6) 29, 7) 30

Таблица 3

Время работы квантового преобразования 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

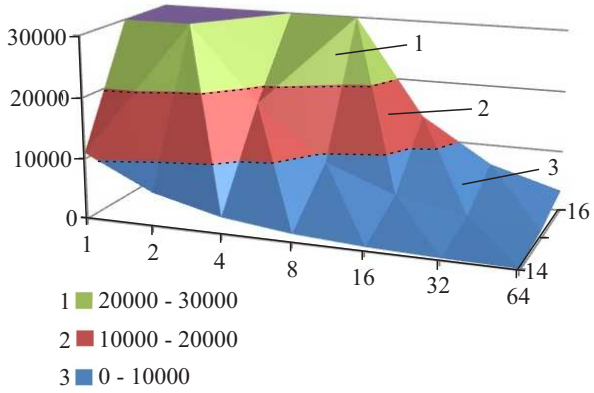

Рис. 7. График времени работы преобразования Гровера в зависимости от количества процессоров и размера задачи: 1) 20000–30000, 2) 10000–20000, 3) 0–10000

ванием 512 вычислительных узлов (4096 процессоров) партиции gpu.

Проводилось моделирование квантового преобразования n-Hadamar. Для моделирования использовались только CPU-узлы. Задача требует большого размера памяти для хранения промежуточных вычислений. Количество кубитов определяет размер обрабатываемых данных, пропорциональный $O\left(2^n\right)$, где n — количество кубитов и количество итераций (равно количеству кубитов). В первом эксперименте увеличивался размер задачи и пропорционально увеличивалось количество ядер. В каждом запуске использовалось 8 ядер на вычислительный узел. В табл. 4 приведены результаты эксперимента. На рис. 8 показан график слабой масштабируемости.

Таблица 4
Результат работы квантового преобразования n-Hadamar

Ядра	Кубиты	Время, сек.	Одна итерация, сек.
4096	38	75.64	1.9905263158
2048	37	71.55	1.9337837838
1024	36	73.06	2.0294444444
512	35	62.63	1.7894285714
256	34	64.03	1.883235294
128	33	60.1	1.8212121212

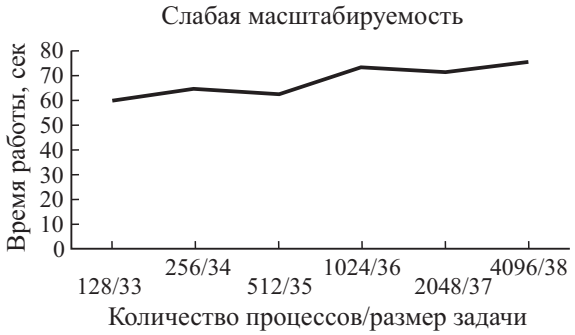

Рис. 8. Время работы программы при пропорциональном увеличении размера задачи и количества ядер (слабая масштабируемость)

Теперь приведем данные для оценки сильной масштабируемости предложенного алгоритма. Таблица 5 содержит время работы программы при увеличении размера задачи для квантового преобразования n-Hadamar. Анализировалось минимально возможное время работы программы при различных размерах задачи на максимально доступном количестве вычислительных узлов. На рис. 9 показан график зависимости времени работы при изменении размера задачи на фиксированном количестве узлов.

Таблица 5
Изменение времени работы программы при увеличении размера задачи для квантового преобразования n-Hadamar

Ядра	Кубиты	Время, сек.
4096	38	75.64
4096	37	34.55
4096	36	18.09
4096	35	8.99
4096	34	4.04
4096	33	1.97
4096	32	1.03
4096	31	0.5
4096	30	0.3

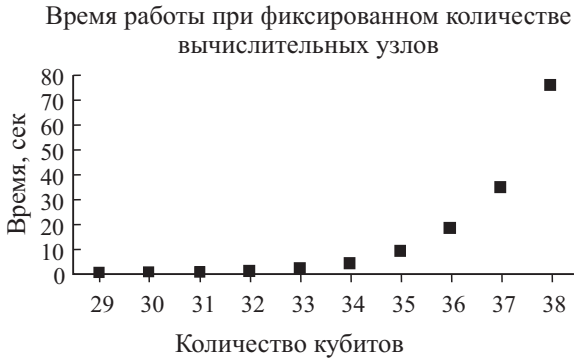

Рис. 9. Зависимость времени работы при изменении размера задачи на фиксированном количестве узлов (512 узлов, 4096 процессов)

6. Заключение. В настоящей статье 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 преобразования Адамара, квантового преобразования Фурье и квантового алгоритма Гровера. Во всех экспериментах показана хорошая масштабируемость разработанной параллельной программы. Дальнейшее развитие работы предполагает проведение моделирования квантовых алгоритмов с шумами.

СПИСОК ЛИТЕРАТУРЫ

1. *Shor P.W.* Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer // SIAM J. Comput. 1997. **26**, N 5. 1484–1509.
2. *Grover L.K.* A fast quantum mechanical algorithm for database search // Proc. of the 28th Annual ACM Symposium on the Theory of Computing. Philadelphia, 1996. 212–219.
3. *Ожигов Ю.И.* Квантовые вычисления. М.: Макс Пресс, 2003.
4. *Нильсен М., Чанг И.* Квантовые вычисления и квантовая информация. М.: Мир, 2006.

5. *Корж А.А.* Масштабирование Data-Intensive приложений с помощью библиотеки DISLIB на суперкомпьютерах Blue Gene/P и "Ломоносов" // Тр. конф. "Научный сервис в сети Интернет-2011". М.: Изд-во Моск. ун-та, 2011. 126–131.
6. *Корж А.А.* Результаты моделирования бенчмарка NBP UA на тысячи ядер суперкомпьютера BlueGene /P с помощью PGAS-расширения OpenMP // Вычислительные методы и программирование. 2010. **11**. 31–41.
7. *Burger J.R.* New approaches to quantum computer simulation in a classical supercomputer // Computing Research Repository (CoRR). 2003. Vol. Quant-ph/0308158.
8. *Tabakin F., Juliá-Díaz B.* QCMPI: A parallel environment for quantum computing // Computer Physics Communications. 2009. N 18. 948–964.
9. *Altschul S., Gish W., Miller W., Myers E., Lipman D.* Basic local alignment search tool // J. of Molecular Biology. 1990. **215 (3)**. 403–410.
10. *Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., du Croz J., Greenbaum A., Hammarling S., McKenney A., Sorensen D.* LAPACK Users' Guide (Third Ed.). Philadelphia: SIAM, 1999.
11. ScaLAPACK (http://www.netlib.org/scalapack/).
12. *Arnold G., Lippert T., Pomplun N., Richter M.* Large Scale Simulation of Ideal Quantum Computers on SMP-Clusters // Proc. of the Conf. on Parallel Computing (ParCo). Malaga, 2005. 447–454.
13. *Negovetic G., Perkowski M., Lukac M., Buller A.* Evolving quantum circuits and an FPGA-based quantum computing emulator // Int. Workshop on Boolean Problems. Freiberg, 2002. 15–22.
14. World record: German supercomputer simulates quantum computer (http://phys.org/news189231849.html).

Поступила в редакцию
15.04.2013