УДК 004.43

Фролов А.С., Семенов А.С.

АО «Научно-исследовательский центр электронной вычислительной техники», г. Москва, Россия

ИССЛЕДОВАНИЕ ПОДХОДОВ К РЕАЛИЗАЦИИ PAGERANK НА ЯЗЫКЕ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ CHARM++*

АННОТАЦИЯ

В статье представлены результаты исследования подходов к реализации задачи ранжирования вершин графа – PageRank, на языке параллельного программирования Charm++, основанном на асинхронной модели вычислений с управлением потоком сообщений. Предложены и реализованы три алгоритма PageRank с разной степенью асинхронности, приведены результаты экспериментов по исследованию производительности реализаций PageRank на 36-узловом вычислительном кластере Ангара-К1 на базе коммуникационной сети Ангара. Для сравнения результатов использовалась реализация PageRank из библиотеки Parallel BGL. Результаты исследования показали, что при небольшом количестве узлов (до 32) на задаче PageRank алгоритмы, разработанные на основе принципов асинхронных вычислений и управления потоком сообщений, но использующие механизмы агрегации коротких сообщений, имеют такую же эффективность, как и алгоритм, использующий классическую BSP-модель.

КЛЮЧЕВЫЕ СЛОВА

Обработка графов, языки параллельного программирования, программные модели, PageRank, супер-компьютеры.

Alexander Frolov, Alexander Semenov

JSC «Scientific and Research Center of Computer Technolohy», Moscow, Russia

APPROACHES TO IMPLEMENTATION OF PAGERANK IN CHARM++

ABSTRACT

In the paper multiple approaches to implementation of the PageRank algorithms in asynchronous, message-driven parallel programming language Charm++ are presented. The described algorithms are based on the native Charm++ asynchronous model, as well as on the aggregation library (known as TRAM) developed for Charm++ applications. Evaluation results have been obtained on the 36-node HPC cluster – Angara K1. A bulk-synchronous parallel PageRank implementation from Parallel BGL has been chosen as a reference for performance comparison. It has been shown that asynchronous algorithms implemented in Charm++ and using a TRAM-based optimization have almost the same performance as the BSP-based PageRank algorithm implemented in Parallel BGL.

KEYWORDS

Graph processing, parallel programming, computation models, PageRank, supercomputers/

ВВЕДЕНИЕ

Одной из прикладных задач теории графов является ранжирование вершин графа, то есть назначение рангов вершинам графа, в соответствии со структурой графа. Типичным применением данной задачи является анализ вэб-графа в поисковых системах [1, 2], однако существуют и другие

* **Труды XI Международной научно-практической конференции «Современные информационные технологии и ИТ-образование» (SITITO'2016), Москва, Россия, 25-26 ноября, 2016**

примеры применения [4, 5, 6]. Одним из первых и наиболее известных алгоритмов ранжирования считается PageRank [7], разработанный для поисковой машины Google. Несмотря на то, что за более чем 15-летнюю историю существования PageRank было выполнено большое количество исследований алгоритма и его модификаций, реализаций с использованием различных программных моделей и технологий, и оптимизаций под различные архитектуры вычислительных систем, классический PageRank продолжает представлять интерес как типовая графовая задача для разработчиков новых программных моделей и инструментальных средств.

В данной работе рассматривается асинхронная вычислительная модель с управлением потоком сообщений и ее реализация в виде языка параллельного программирования Charm++ [8, 9] для суперкомпьютерных комплексов и вычислительных кластерных систем. Программная модель Charm++ значительно отличается от существующих стандартов разработки параллельных программ, основанных на SPMD-модели и модели распределенной или общей памяти, MPI [10] или OpenMP [11], а также их комбинаций. Асинхронность? заложенная в модели вычислений Charm++? позволяет по-другому выражать алгоритмы и, соответственно, получать принципиально иную организацию вычислительного процесса. Более подробно о существующих асинхронных моделях вычислений можно ознакомится в работе [12].

Алгоритмы анализа графов на Charm++ являются малоисследованной областью до настоящего времени, в то время как модель Charm++ хорошо подходит для разработки асинхронных графовых алгоритмов в силу следующих причин:

- объектно-ориентированная модель Charm++ позволяет достаточно просто выражать графовые алгоритмы в вершинно-ориентированном стиле (*vertex-centric*), который является одним из наиболее распространенных подходов к работе с графами при решении задач анализа Больших Данных [13, 14];

- динамическая балансировка нагрузки, поддерживаемая в системе времени выполнения (runtime-системе) языка Charm++, позволяет осуществлять автоматическую миграцию объектов между узлами вычислительного кластера, что особенно актуально для графовых задач вследствие их нерегулярности, а также несбалансированности реальных графов (например, *scale-free* графы с распределением степени вершин по степенному закону).

В данной работе представлены три различных алгоритма реализации PageRank на Charm++, все три алгоритма являются асинхронными по характеру выполнения модификаций рангов вершин, но с различной степенью асинхронности. Реализации алгоритмов были протестированы на синтетически сгенерированных графах типа RMAT [15]; данный тип близок к графам, используемым в оценочном тесте Graph500 [16]. Результаты оценочного тестирования получены на вычислительном кластере Ангара-К1 на базе коммуникационной сети Ангара [17,18].

Далее приводится краткое описание программной модели Charm++ (раздел 2), описание формальной постановки задачи PageRank (раздел 3), описание алгоритмов (раздел 4) и анализ результатов оценочного тестирования (раздел 5). В заключении приведены выводы и дальнейшие планы исследований.

ПРОГРАММНАЯ МОДЕЛЬ CHARM++

Язык параллельного программирования Charm++ является расширением языка C++ и основан на объектно-ориентированной модели с управлением асинхронным потоком сообщений [8, 9]. Ниже приводятся основные концепции программной модели Charm++.

Базовым объектом в Charm++ является *chare* (или *chare-объект*), обладающий помимо всех свойств объектов в C++ (инкапсулированными данными, множеством публичных и приватных методов и т. п.), интерфейсом из специальных *entry-методов*, вызовы которых соответствуют передаче данных (т. е. сообщений) между chare-объектами.

Приложение в Charm++ состоит из множества chare-объектов, обменивающихся между собой данными посредством вызовов entry-методов. Кроме того, вызов entry-метода порождает выполнение непосредственно самого метода на том узле, где находится chare-объект, entry-метод которого был вызван, то есть таким образом реализуется концепция *управления потоком данных* или *активных сообщений*.

В процессе выполнения entry-метода могут быть изменены только данные, принадлежащие chare-объекту. Одновременно у chare-объекта может выполняться не более одного entry-метода, что позволяет избежать проблемы обеспечения атомарности изменения данных, принадлежащих chare-объекту, и значительно упрощает программирование. Схематически программная модель Charm++ представлена на рисунке 1.

Кроме простых chare-объектов Charm++ имеется возможность создавать массивы chare-объектов. При этом массивы бывают как одномерными, так и многомерными. Использование

массивов позволяет создавать большое количество chare-объектов, управлять их распределением по вычислительным узлам, выполнять над ними коллективные операции.

В Charm++ поддерживается автоматическая динамическая балансировка нагрузки на вычислительные узлы, что реализуется на уровне runtime-системы за счет перемещения chare-объектов между вычислительными узлами. Для прикладного программиста данная возможность совершенна прозрачна, так как программная модель Charm++ не специфицирует расположение chare-объектов в вычислительной системе, и все объекты адресуются в едином адресном пространстве.

Основные концепции модели Charm++ были разработаны почти 20 лет назад в то время, когда массово-параллельные системы (вычислительные кластеры) на основе коммерческих микропроцессоров только набирали популярность. Тем не менее, модель Charm++, вобравшая в себя как принципы dataflow-моделей, так и многопоточных (мультитредовых) моделей вычислений, оказалась достаточно успешной и с момента создания почти не претерпевала изменений.

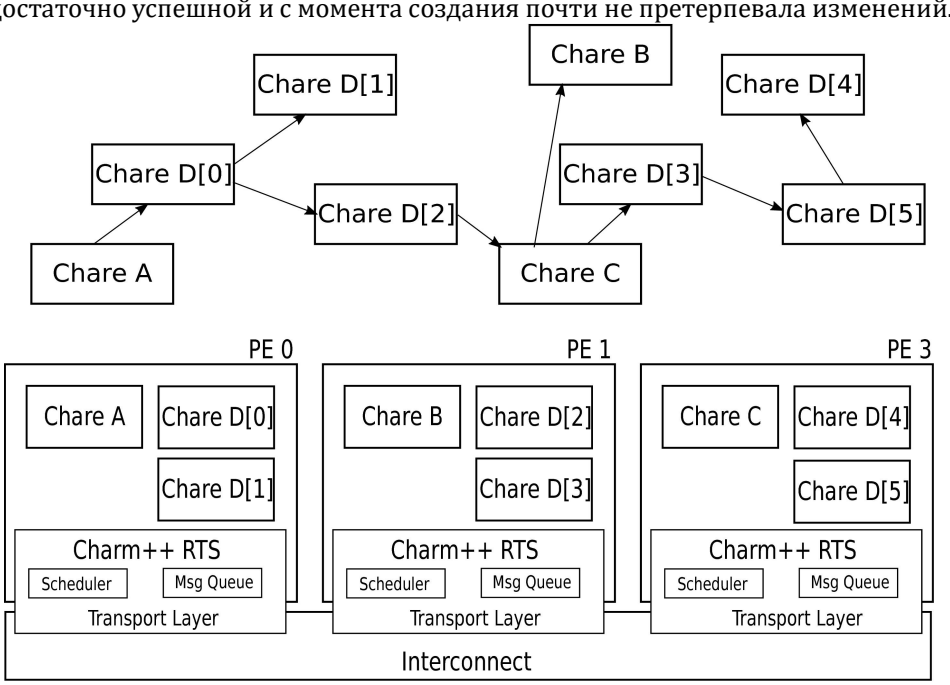

Рис. 1. Программная модель Charm++

ПОСТАНОВКА ЗАДАЧИ PAGERANK

Алгоритм PageRank моделирует процесс случайного перемещения ("серфинга") пользователя по страницам вэб-сети, при этом перед каждым новым шагом с заданной вероятностью выбирается одно из двух действий: (1) пользователь с равной вероятностью выбирает одну из гиперссылок (ребро вэб-графа), принадлежащих текущей странице (вершине вэб-графа), и переходит на страницу по выбранной ссылке, или (2) пользователь осуществляет "телепортацию" – перемещение на другую страницу, выбранную случайным образом из всех существующих страниц вэб-сети. Интуитивно понятно, что при таком перемещении, чем больше страница имеет входящих ссылок, тем большее количество раз на нее будет выполнен переход.

Пусть дан ориентированный граф$G = (V, E)$, в котором V– это множество вершин, E– множество дуг. Тогда ранг вершины vв PageRank вычисляется следующим образом:

$$PR(v) = \frac{1-p}{|V|} + p \times \sum_{u \in d_{\text{in}}(v)} \frac{PR(u)}{|d_{out}(u)|}, \quad (1)$$

где p– вероятность выбора перехода по исходящей ссылке $(0 < p < 1)$, $d_{\text{in}}(v)$– множество смежных с vвершин, имеющих исходящие дуги, указывающие на v, $d_{\text{out}}(v)$– множество смежных с vвершин, на которые указывают исходящие из vдуги. В качестве начального значения рангов выбирается $\frac{1}{|V|}$.

АСИНХРОННЫЕ АЛГОРИТМЫ PAGERANK

Наиболее распространенной схемой вычисления рангов вершин при реализации алгоритма PageRank является итеративная схема, когда на каждом шаге вычисляется новое значение ранга для каждой вершины, при этом между шагами, как правило выполняется барьерная синхронизация, чтобы обеспечить согласованность вычислений. В нашем подходе мы также использовали итерационную схему вычислений, однако адаптировали ее к асинхронной модели Charm++.

На рисунке 2 представлен базовый (или наивный) асинхронный алгоритм PageRank на языке Charm++. Алгоритм является итерационным, на каждой итерации происходит пересчет значений рангов всех вершин. Вершины представлены в виде элементов массива chare-объектов класса PageRankVertex, при этом каждый объект соответствует одной вершине графа. Далее термины вершина и chare-объект можно считать взаимозаменяемыми.

```
readonly:
g – прокси-ссылка на массив chare-объектов (граф)
N – количество вершин
d – вероятность перехода по ребру (damping factor)
entry method PageRankVertex:: doPageRankStep_init()
    PR_old ← PR_new
    PR_new ← (1.0 − d)/N
end method
entry method PageRankVertex::doPageRankStep_update()
    ▷ вызов update в соседних вершинах
    for all u ∈ outgoing edges do
        g[dest(u)].update(PR_old/d_out)
    end for
end method
entry method PageRankVertex::update(r)
    PR_new ← PR_new + d × r
end method
function MAINCHARE::DOPAGERANK
    for i = 0; i < N_iters; i ← i + 1 do
        ▷ вызов doPageRankStep_init во всех вершинах графа
        g[∗].doPageRankStep_init()
        ▷ ожидание “тишины”
        CkStartQD(CkCallbackResumeThread())
        ▷ вызов doPageRankStep_update во всех вершинах графа
        g[∗].doPageRankStep_update()
        ▷ ожидание “тишины”
        CkStartQD(CkCallbackResumeThread())
    end for
end function
```

Рис. 2. Базовый алгоритм PageRank на Charm++

В каждой вершине определены два параметраPR_{old}иPR_{new}, в которых хранятся значение ранга вершины для предыдущей итерации и частично подсчитанное значение ранга на текущей итерации, соответственно. В начальный момент времениPR_{new}инициализируется значением$\frac{1}{|V|}$.

Итерации алгоритма состоят из двух фаз. Первая фаза инициируется вызовом entry-метода doPageRankStep_init у всех вершин. Назначение первой фазы в инициализации текущего шага – обновлении значенийPR_{old}иPR_{new}. Вторая фаза инициируется вызовом doPageRankStep_update также у всех вершин. На второй фазе выполняется асинхронное вычислениеPR_{new}у всех вершин. При этом каждая вершина рассылает своим смежным вершинам (по исходящим дугам) значение$\frac{PR_{old}}{d_{out}}$с помощью вызова entry-метода update у соответствующих chare-объектов. Между фазами, так же как и между итерациями, необходимо выполнение барьерной синхронизации. Для этого используется поддерживаемый в Charm++ механизм обнаружения "тишины" (вызов CkStartQD).

Очевидным недостатком базового алгоритма PageRank является наличие двух глобальных барьерных синхронизаций на каждой итерации, что может привести к слабой масштабируемости алгоритма. Для устранения этой проблемы предлагаются следующие два алгоритма: частично-асинхронный алгоритм PageRank с подсчетом сообщений и полностью асинхронный алгоритм.

На рисунке 3 приведен псевдокод частично-асинхронного алгоритма PageRank с подсчетом входящих сообщений, в котором количество глобальных барьеров на одну итерацию сокращено до одного барьера. Основная идея алгоритма – использовать подсчет входящих сообщений (вызовов entry-метода update), и после получения последнего сообщения на данной итерации обновлять значения локальных переменных вершины. Подразумевается, что все вершины имеют информацию о количестве входящих соседей (d_{in}), так что для ориентированных графов требуется дополнительная предобработка графа. Для подсчета сообщений используется обратный счетчик n_{msg}.

```
readonly:
g – прокси-ссылка на массив chare-объектов (граф)
N – количество вершин
d – вероятность перехода по ребру (damping factor)
entry method PageRankVertex::doPageRankStep()
    ▷ присваивание PR_old ссылки на rank_0 или rank_1
    PR_old ↤ (n_iter mod 2) ? rank_0 : rank_1
    ▷ вызов update в соседний вершины
    for all u ∈ outgoing edges do
        g[dest(u)].update(PR_old/d_out)
    end for
end method
entry method PageRankVertex::update(r)
    ▷ присваивание PR_new ссылки на rank_0 или rank_1
    PR_new ↤ (n_iter mod 2) ? rank_1 : rank_0
    PR_new ← PR_new + d × r
    n_msg ← n_msg − 1
    if n_msg = 0 then
        n_msg ← d_in
        n_iter ← n_iter + 1
        ▷ присваивание PR_new ссылки на rank_0 или rank_1
        PR_new ↤ (n_iter mod 2) ? rank_1 : rank_0
        PR_new ← (1.0 − d)/N
    end if
end method
function TestDriver::doPageRank
    for i = 0; i < N_iters; i ← i + 1 do
        ▷ вызов doPageRankStep во всех вершинах графа
        g[∗].doPageRankStep()
        ▷ ожидание "тишины"
        CkStartQD(CkCallbackResumeThread())
    end for
end function
```

Рис. 3. Алгоритм PageRank с подсчетом сообщений на Charm++

```
readonly:
g – прокси-ссылка на массив chare-объектов (граф)
N – количество вершин
d – вероятность перехода по ребру (damping factor)
N_iters – количество итераций PageRank
entry method PageRankVertex::doPageRank()
    ▷ инициализация PR_cur и iter в каждой вершине
    PR_cur ← 1.0/N, iter ← 0
    ▷ вызов update в соседних вершинах
    for all u ∈ outgoing edges do
        g[dest(u)].update(0, PR_cur/d_out)
    end for
    iter ← iter + 1
end method
entry method PageRankVertex::update(i, r)
    PR[i].n_msg ← PR[i].n_msg + 1
    PR[i].pr ← PR[i].pr + d × r
    while PR[iter].n_msg = d_in do
        PR_cur = (1.0 − d)/N + PR[iter].pr
        if iter + 1 < N_iters then
            ▷ вызов update в соседних вершинах
            for all u ∈ outgoing edges do
                g[dest(u)].update(i + 1, PR_cur/d_out)
            end for
            iter ← iter + 1
        end if
    end while
end method
function MainChare::doPageRank
    ▷ вызов doPageRank во всех вершинах
    g[∗].doPageRank()
end function
```

Рис. 4. Полностью асинхронный алгоритм PageRank на Charm++

Также необходимо обратить внимание, что работа с переменными хранящими предыдущее и текущее (обновляемое) значения рангов, выполняется через ссылки (PR_{old}и PR_{new}), это необходимо для того, чтобы сохранить консистентность этих значений в случае рассинхронизации выполнения методов doPageRank и update.

На рисунке 4 представлен еще одна модификация алгоритма PageRank с полностью асинхронным выполнением обновлений рангов вершин, то есть между итерациями PageRank отсутствует какая-либо синхронизация. Для этого в каждую вершину добавлена ассоциативная таблица$PR[\ ]$, адресуемая по номеру итерации, а в вызов entry-метода update добавлен номер итерации (i), в рамках которой он выполняется.

Каждая запись $PR[\ ]$содержит значение ранга вершины на данной итерации pr, которое может быть вычисленным или быть в стадии вычисления, и количество сообщений n_{msg}, полученных на данной итерации от соседних вершин. Когда все сообщения были доставлены для текущей итерации (т. е. $PR[iter].n_{msg} = d_{\text{in}}$), то вершина "переходит" на следующую итерацию ($iter = iter + 1$), рассылает новые значения своим соседям и ожидает новую порцию сообщений и т. д. Необходимо отметить, что в вершину могут приходить сообщения с разными i, и тогда информация о них будет попадать в разные записи $PR[\ ]$.

Экспериментальные результаты

В качестве тестовой платформы использовался вычислительный кластер Ангара-К1 с отечественной коммуникационной сетью Ангара. Конфигурация Ангара-К1 включает: (1) 24 двухпроцессорных узла с 6-ядерными процессорами Intel Xeon E5-2630, (2) 12 однопроцессорных узла с 8-ядерными Intel Xeon E5-2660. Все узлы имеют по 64 ГБ оперативной памяти. Коммуникационная сеть – Ангара с топологией 3D-тор (3x3x4).

Для тестирования масштабируемости алгоритмов использовались ориентированные синтетические RMAT графы различного размера (параметр k=16). В качестве генератора использовался соответствующий код из тестового пакета Graph500. Количество итераций PageRank равно 3.

На рисунке 5 приведены графики зависимостей времени выполнения PageRank от количества используемых узлов кластера (количество используемых ядер в узле было всегда равным 8) при постоянном размере графа (сильная масштабируемость). Таким образом, число вершин графа RMAT-s составляет2^s.

Из графиков видно, что, во-первых, базовый алгоритм PageRank и алгоритм с подсчетом сообщений показывают практически одинаковое время выполнения. С одной стороны, это говорит о достаточно эффективной реализации механизма обнаружения тишины в Charm++, но, с другой стороны, максимальное количество узлов достаточно мало и негативный эффект от выполнения барьерной синхронизации не заметен. Во-вторых, полностью асинхронный алгоритм PageRank имеет более высокую производительность по сравнению с базовым алгоритмом: так, для графа RMAT-20, на одном узле отношение двух алгоритмов составляет 2,5 раза, а на 16 узлах – 3,1 раза. Данное ускорение может быть объяснено несбалансированностью вычислений в рамках одной итерации при выполнении базового алгоритма PageRank (также как и алгоритма с подсчетом сообщений), в то время как в асинхронном алгоритме данная несбалансированность не имеет сильного значения, так как итерации выполняются с перекрытием друг относительно друга.

Также была выполнена реализация базового алгоритма с использованием библиотеки агрегации сообщений TRAM [19], разработанной для Charm++ программ. Применение этой библиотеки позволяет отправлять короткие сообщения, используя механизмы агрегации и программной маршрутизации сообщений в заданной виртуальной топологии (поддерживаются многомерные решетки), в узлах которой находятся процессы системы поддержки выполнения Charm++ (или, что практически одно и то же, MPI-процессы).

Как видно из рисунка 5 использование библиотеки TRAM для реализации базового алгоритма позволило значительно сократить время выполнения алгоритма по сравнению с нативной Charm++ реализации: для графа RMAT-20 на одном узле прирост производительности составил 4,22 раза, а на 16 узлах – 5,7 раза. Таким образом, реализация базового алгоритма PageRank с использованием библиотеки TRAM оказалась наиболее эффективной среди исследуемых реализаций PageRank на Charm++.

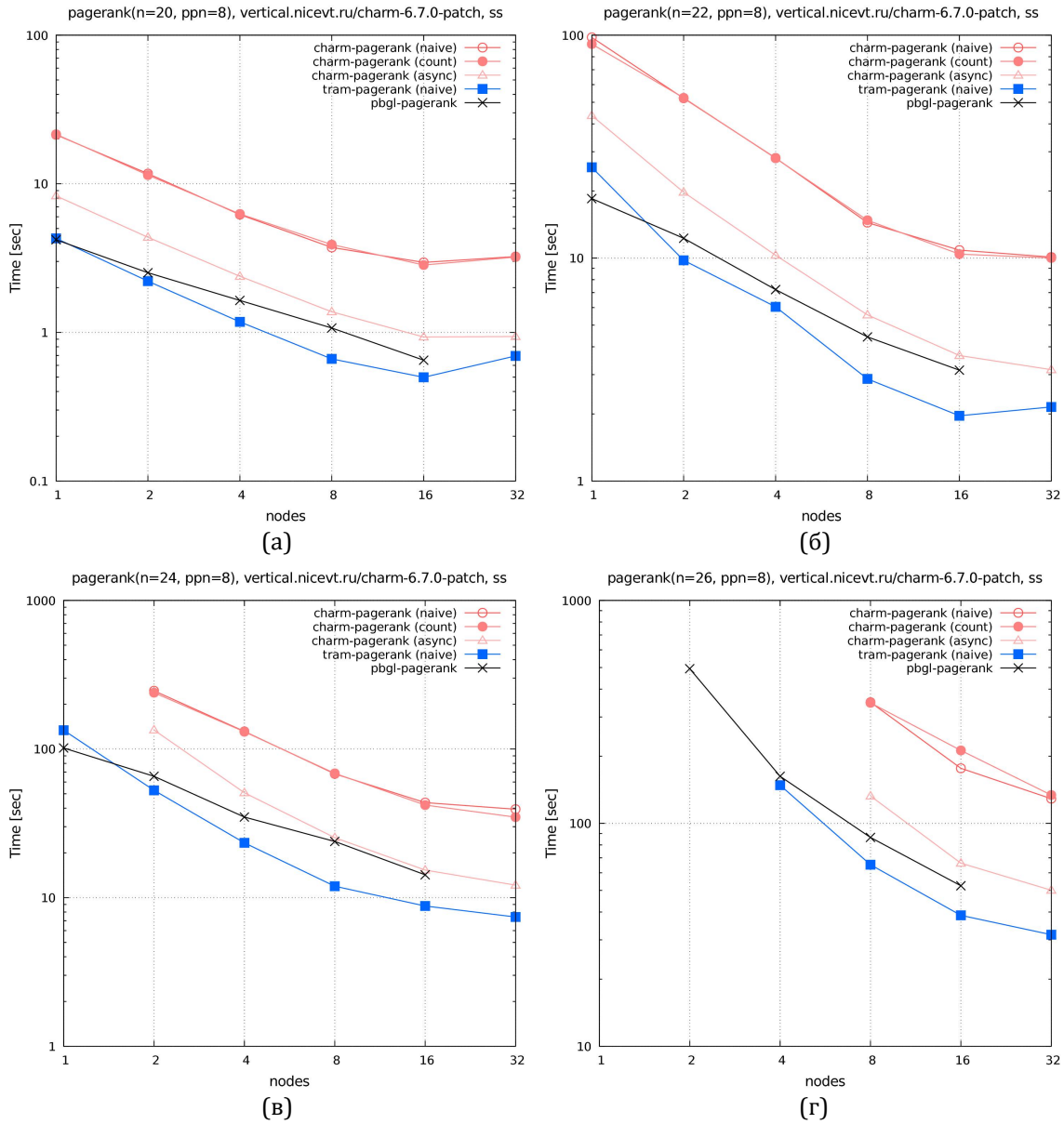

Рис. 5. Время выполнения PageRank на кластере Ангара-К1 для графов RMAT-20 (а), RMAT-22 (б), RMAT-24 (в), RMAT-26 (г). Режим сильной масштабируемости

Для сравнения были получены результаты выполнения алгоритма PageRank, реализованного в библиотеке Parallel Boost Graph Library (PBGL) [20], основанной на синхронной модели вычислений Bulk-Synchronous Parallel [21]. Как видно из графиков, что PBGL и TRAM реализации PageRank имеет практически одинаковое время выполнения, что объясняется схожими принципами их работы: агрегация коротких сообщений и совмещение вычислений и передачи данных по сети.

Выколотые точки на графиках (графы RMAT-24 и RMAT-26) соответствуют случаям, когда выполнение алгоритмов невозможно из-за нехватки памяти. При этом в асинхронных алгоритмах проблема нехватки памяти особенно актуальна, так как для таких алгоритмов характерно резкое возрастание количества сообщений и, соответственно, аллокация больших участков в памяти (проблема взрывного параллелизма).

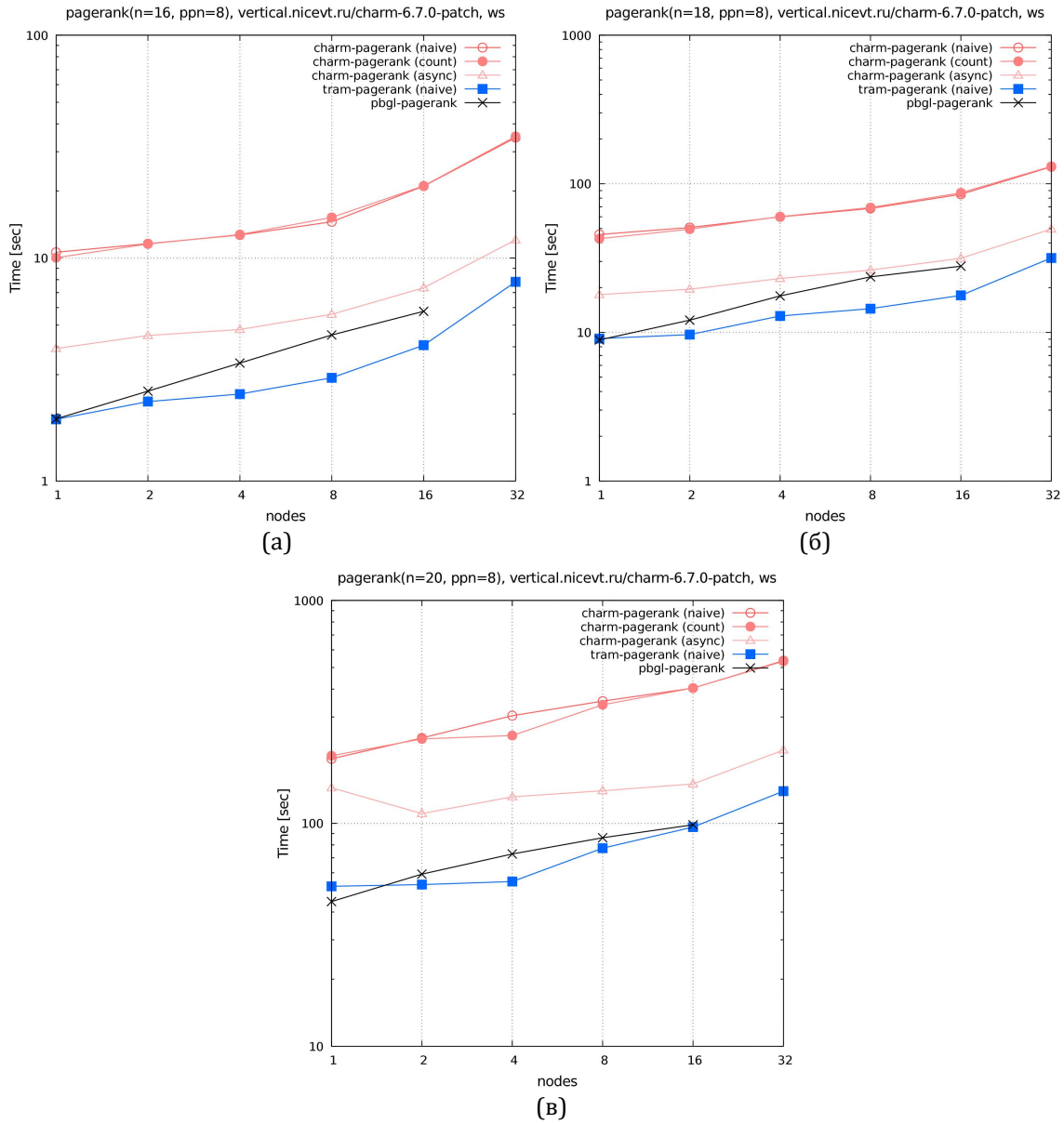

Рис. 6. Время выполнения PageRank на кластере Ангара-К1 для графов RMAT-16 (а), RMAT-18 (б) и RMAT-20 (в). Режим слабой масштабируемости

На рисунке 6 приведены графики зависимости времени выполнения PageRank от количества используемых узлов кластера (количество используемых ядер в узле – 8) при размере графа с пропорциональным увеличением от числа узлов (слабая масштабируемость). Таким образом, количество вершин графа RMAT-s составляет2^{s*8*n}, где n– количество используемых узлов.

Как видно из графиков на рисунке 6, все эффекты, наблюдаемые в случае сильного масштабирования, проявляются и при слабом масштабировании. Отличие лишь в несколько большей разнице в производительности между асинхронным PageRank, реализованном с использованием TRAM, и реализацией на PBGL (особенно для графа RMAT-18).

Заключение

В статье представлены три асинхронных алгоритма PageRank, реализованных на языке параллельного программирования Charm++, а также с использованием библиотеки TRAM. Исследование производительности алгоритмов проводилось на 36-узловом вычислительном кластере Ангара-К1 на базе коммуникационной сети Ангара.

Результаты экспериментов показали, что среди реализаций? выполненных на Charm++ наиболее эффективным оказался полностью асинхронный алгоритм, в котором итерации PageRank выполняются одновременно. Применение библиотеки TRAM для реализации базового алгоритма позволило получить почти 5-кратное увеличение производительности. В сравнении с

существующей реализацией PageRank в параллельной библиотеке PBGL реализация PageRank на TRAM показывает незначительно большую эффективность.

В данной работе на примере задачи PageRank продемонстрировано, что с помощью асинхронной модели вычислений Charm++ (с использованием TRAM) возможно разрабатывать и реализовывать графовые алгоритмы в вершинно-ориентированном стиле (т. е. стиле *vertex-centric*), при этом производительность таких алгоритмом сопоставима с производительностью решений на базе традиционных моделей вычислений (в данной работе – BSP модели и ее реализации в библиотеке PBGL).

В будущем предполагается сравнение Charm++ с другими системами, поддерживающими асинхронную модель вычислений, такими как AM++ (ActivePebbles) [22], Grappa [23], HPX [24], а также исследование масштабируемости алгоритмов на графах большего размера (RMAT-30 и больше), для чего потребуются кластерные системы с большим числом узлов.

Работа выполнена при поддержке гранта РФФИ №15-07-09368.

Литература

1. Brin S., Page L. Reprint of: The anatomy of a large-scale hypertextual web search engine //Computer networks. – 2012. – Т. 56. – №. 18. – С. 3825-3833.
2. Langville A. N., Meyer C. D. Google's PageRank and beyond: The science of search engine rankings. – Princeton University Press, 2011.
3. Suri P. R., Taneja H. An integrated ranking algorithm for efficient information computing in social networks //arXiv preprint arXiv:1204.1413. – 2012.
4. Sharma D. et al. A ranking algorithm for online social network search //Proceedings of the 6th ACM India Computing Convention. – ACM, 2013. – С. 17.
5. Winter C. et al. Google goes cancer: improving outcome prediction for cancer patients by network-based ranking of marker genes //PLoS Comput Biol. – 2012. – Т. 8. – №. 5..
6. Morrison J. L. et al. GeneRank: using search engine technology for the analysis of microarray experiments //BMC bioinformatics. – 2005. – Т. 6. – №. 1. – С. 1.
7. Page L. et al. The PageRank citation ranking: bringing order to the web. – 1999.
8. Kale L. V., Krishnan S. CHARM++: a portable concurrent object oriented system based on C++. – ACM, 1993. – Т. 28. – №. 10. – С. 91-108.
9. Acun B. et al. Parallel programming with migratable objects: charm++ in practice //SC14: International Conference for High Performance Computing, Networking, Storage and Analysis. – IEEE, 2014. – С. 647-658.
10. Official MPI Forum web-site, http://mpi-forum.org
11. Official OpenMP web-site, http://openmp.org
12. Фролов А.С., Семенов А.С., Марков А.С. Обзор инструментальных средств разработки параллельных графовых приложений для суперкомпьютерных комплексов // «Вычислительные нанотехнологии». – №4. – 2015.
13. McCune R. R., Weninger T., Madey G. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing //ACM Computing Surveys (CSUR). – 2015. – Т. 48. – №. 2. – С. 25.
14. Malewicz G. et al. Pregel: a system for large-scale graph processing //Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. – ACM, 2010. – С. 135-146.
15. Seshadhri C., Pinar A., Kolda T. G. An in-depth study of stochastic Kronecker graphs //2011 IEEE 11th International Conference on Data Mining. – IEEE, 2011. – С. 587-596.
16. Murphy R. C. et al. Introducing the graph 500 //Cray User's Group (CUG). – 2010.
17. Симонов А.С., Жабин И.А., Макагон Д.В. Разработка межузловой коммуникационной сети с топологией «многомерный тор» и поддержкой глобально адресуемой памяти для перспективных отечественных суперкомпьютеров. // Научно-техническая конференция «Перспективные направления развития вычислительной техники», ОАО «НИЦЭВТ», 2011.
18. Симонов А.С., Макагон Д.В., Жабин И.А., Щербак А.Н., Сыромятников Е.Л., Поляков Д.А. Первое поколение высокоскоростной коммуникационной сети «Ангара» // Наукоемкие технологии. — 2014. — Т. 15, No1. — С. 21-28.
19. Wesolowski L. et al. Tram: Optimizing fine-grained communication with topological routing and aggregation of messages //2014 43rd International Conference on Parallel Processing. – IEEE, 2014. – С. 211-220.
20. Gregor D., Lumsdaine A. The parallel BGL: A generic library for distributed graph computations //Parallel Object-Oriented Scientific Computing (POOSC). – 2005. – Т. 2. – С. 1-18.
21. Cheatham T. et al. Bulk synchronous parallel computing—a paradigm for transportable software //Tools and Environments for Parallel and Distributed Systems. – Springer US, 1996. – С. 61-76.
22. Willcock J. J. et al. Active pebbles: parallel programming for data-driven applications //Proceedings of the international conference on Supercomputing. – ACM, 2011. – С. 235-244.
23. Nelson J. et al. Grappa: A latency-tolerant runtime for large-scale irregular applications //International Workshop on Rack-Scale Computing (WRSC w/EuroSys). – 2014.
24. Kaiser H., Brodowicz M., Sterling T. Parallex an advanced parallel execution model for scaling-impaired applications //2009 International Conference on Parallel Processing Workshops. – IEEE, 2009. – С. 394-401.

References

1. Brin S., Page L. Reprint of: The anatomy of a large-scale hypertextual web search engine //Computer networks. – 2012. – Т. 56. – №. 18. – С. 3825-3833.
2. Langville A. N., Meyer C. D. Google's PageRank and beyond: The science of search engine rankings. – Princeton University Press, 2011.

3. Suri P. R., Taneja H. An integrated ranking algorithm for efficient information computing in social networks //arXiv preprint arXiv:1204.1413. – 2012.
4. Sharma D. et al. A ranking algorithm for online social network search //Proceedings of the 6th ACM India Computing Convention. – ACM, 2013. – С. 17.
5. Winter C. et al. Google goes cancer: improving outcome prediction for cancer patients by network-based ranking of marker genes //PLoS Comput Biol. – 2012. – Т. 8. – №. 5..
6. Morrison J. L. et al. GeneRank: using search engine technology for the analysis of microarray experiments //BMC bioinformatics. – 2005. – Т. 6. – №. 1. – С. 1.
7. Page L. et al. The PageRank citation ranking: bringing order to the web. – 1999.
8. Kale L. V., Krishnan S. CHARM++: a portable concurrent object oriented system based on C++. – ACM, 1993. – Т. 28. – №. 10. – С. 91-108.
9. Acun B. et al. Parallel programming with migratable objects: charm++ in practice //SC14: International Conference for High Performance Computing, Networking, Storage and Analysis. – IEEE, 2014. – С. 647-658.
10. Official MPI Forum web-site, http://mpi-forum.org
11. Official OpenMP web-site, http://openmp.org
12. Frolov A.S., Semenov A.S., Markov A.S. Obzor instrumental'nykh sredstv razrabotki parallel'nykh grafovykh prilozheniy dlya superkomp'yuternykh kompleksov // «Vychislitel'nye nanotekhnologii». – №4. – 2015.
13. McCune R. R., Weninger T., Madey G. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing //ACM Computing Surveys (CSUR). – 2015. – Т. 48. – №. 2. – С. 25.
14. Malewicz G. et al. Pregel: a system for large-scale graph processing //Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. – ACM, 2010. – С. 135-146.
15. Seshadhri C., Pinar A., Kolda T. G. An in-depth study of stochastic Kronecker graphs //2011 IEEE 11th International Conference on Data Mining. – IEEE, 2011. – С. 587-596.
16. Murphy R. C. et al. Introducing the graph 500 //Cray User's Group (CUG). – 2010.
17. Simonov A.S., Zhabin I.A., Makagon D.V. Razrabotka mezhuzlovoy kommunikatsionnoy seti s topologiey «mnogomernyy tor» i podderzhkoy global'no adresuemoy pamyati dlya perspektivnykh otechestvennykh superkomp'yuterov. // Nauchno-tekhnicheskaya konferentsiya «Perspektivnye napravleniya razvitiya vychislitel'noy tekhniki», OAO «NITsEVT», 2011.
18. Simonov A.S., Makagon D.V., Zhabin I.A., Shcherbak A.N., Syromyatnikov E.L., Polyakov D.A. Pervoe pokolenie vysokoskorostnoy kommunikatsionnoy seti «Angara» // Naukoemkie tekhnologii. — 2014. — T. 15, No1. — S. 21-28.
19. Wesolowski L. et al. Tram: Optimizing fine-grained communication with topological routing and aggregation of messages //2014 43rd International Conference on Parallel Processing. – IEEE, 2014. – С. 211-220.
20. Gregor D., Lumsdaine A. The parallel BGL: A generic library for distributed graph computations //Parallel Object-Oriented Scientific Computing (POOSC). – 2005. – Т. 2. – С. 1-18.
21. Cheatham T. et al. Bulk synchronous parallel computing—a paradigm for transportable software //Tools and Environments for Parallel and Distributed Systems. – Springer US, 1996. – С. 61-76.
22. Willcock J. J. et al. Active pebbles: parallel programming for data-driven applications //Proceedings of the international conference on Supercomputing. – ACM, 2011. – С. 235-244.
23. Nelson J. et al. Grappa: A latency-tolerant runtime for large-scale irregular applications //International Workshop on Rack-Scale Computing (WRSC w/EuroSys). – 2014.
24. Kaiser H., Brodowicz M., Sterling T. Parallex an advanced parallel execution model for scaling-impaired applications //2009 International Conference on Parallel Processing Workshops. – IEEE, 2009. – С. 394-401.

Поступила: 10.10.2016

Об авторах:

Фролов Александр Сергеевич, начальник отдела, АО «Научно-исследовательский центр электронной вычислительной техники» (АО «НИЦЭВТ»), frolov@nicevt.ru;

Семенов Александр Сергеевич, кандитат технических наук, начальник сектора, АО «Научно-исследовательский центр электронной вычислительной техники» (АО «НИЦЭВТ»), semenov@nicevt.ru.