Эффективность распараллеливания явных формул для подсчета коротких циклов в графе

А.М. Караваев, А.Н. Воропаев

Созданы бинарные реализации формул, недавно полученных авторами в системе компьютерной алгебры. Рассматриваются выражения для циклов длиной до 12 в произвольных графах и циклов длиной до 14 в двудольных графах. Аддитивная структура формул позволяет при их распараллеливании достигнуть линейного по количеству процессоров ускорения и проводить вычисления на графах с несколькими сотнями вершин. Учет двудольности графа в формулах значительно сокращает время вычисления по ним.

1. Введение

Подсчет простых циклов представляет интерес в различных областях науки. Одна из них — исследование структурных свойств реальных сетей (социальные сети, «Всемирная паутина», схемы городских улиц) [1–4]. Количество циклов также изучают при исследовании решеточных моделей статистической механики и кодов с исправлением ошибок [5].

В данной работе используется, в основном, терминология [6].

Задача подсчета количества c_k простых циклов длиной k в графе [7, с. 267] в общем случае (для произвольных значения k и графа) труднорешаема, как ее частный вариант — проблема гамильтоновых циклов [7, с. 266], [8, с. 209]. Более того, в [9] показано, что эта задача $\#W[1]$-полна, когда параметром является длина цикла, то есть не существует $f(k)\,n^{\mathrm{const}}$-алгоритма вычисления c_k, где n — порядок графа, если $\#W[1] \neq FPT$.

Известно [10–12], что количество простых циклов длиной k вычисляется со сложностью умножения матриц при $k \leqslant 7$ и на порядок сложнее при $k = 8$. Авторы [13] сообщили о явных формулах для величин c_k, вычислительная сложность которых при $k \geqslant 8$ имеет оценку $O(n^{[k/2]} \log n)$, а в случае двудольных графов для $k = 8, 10, 14$ — на порядок меньше. Сами выражения выведены только при $k \leqslant 12$ для произвольных графов и $k \leqslant 14$ для двудольных графов. Хотя результаты [13] представляют собой явные формулы, уже выражение для c_{10} в случае произвольных графов при нескольких часах вычислений в системе компьютерной алгебры на ПК справляется только с графами, имеющими до 70 вершин.

Альтернативный алгоритм подсчета циклов заданной длины в произвольных графах — по явной формуле [14] — не ограничен малыми значениями k, но имеет экспоненциальную сложность. В [15] и [13] получены упрощенные варианты этой формулы с вычислительной сложностью $O(n^{k-1})$ и $O(n^k)$ при фиксированной длине k, что вдвое больше по порядку, чем сложность указанных выше выражений [13]. За несколько часов вычислений по формуле [15] в системе компьютерной алгебры на ПК значение c_{10} удается получить при числе вершин до 40. В общем случае оказывается неэффективным и подсчет циклов путем их перечисления какими бы то ни было алгоритмами, например, [16–24], так как само количество простых циклов фиксированной длины k в графе порядка n может расти со скоростью $O(n^k)$.

Возможности параллельного вычисления количества циклов широко не обсуждаются. В [25, 26] выведены формулы для подсчета гамильтоновых путей и контуров, всех путей и контуров, всех контуров с заданной вершиной и всех путей, соединяющих заданные вершины. Ввиду экспоненциальной сложности эти выражения практически применимы для графов порядком в несколько десятков. Автор названных работ только указал, что полученные формулы хорошо подходят для параллельной реализации. Аналогичное замечание имеется в [24], где предложен алгоритм перечисления всех контуров или контуров до заданной длины, основанный на обходе орграфа в ширину. Результаты [25] частично воспроизводят ранее известные формулы [14, 27], которые имеют аналогичную аддитивную структуру и так же хорошо распараллеливаемы. Авторы [28] разработали алгоритм перечисления всех

циклов планарного графа на основе пространства циклов в последовательном и параллельном (EREW PRAM) вариантах и реализовали его на параллельном SIMD-компьютере с матричными соединениями (mesh-connected) MasPar MP-1 с 1024 узлами. Вычислительные эксперименты показали, что достигается $O(p)$-ускорение, где p — количество процессоров. Абсолютное время работы программы в [28] не приводится.

Упоминавшиеся выше явные формулы [13] для подсчета коротких циклов в произвольных и двудольных графах наиболее эффективны из рассмотренных алгоритмов и ввиду аддитивной структуры пригодны для параллельной реализации. Однако имеется их воплощение только в системе компьютерной алгебры. Настоящая работа выполнялась с двумя основными целями.

- Создать параллельные бинарные реализации выражений.
- Практически оценить эффективность распараллеливания формул и порядки графов, с которыми параллельные реализации выражений справляются за небольшое время.

Круг задач, которые требовалось выполнить, включал

- перевод формул с языка системы компьютерной алгебры на языки общего назначения (C и C++) с параллельными расширениями, так как встроенные средства экспорта не обладают вполне достаточными для этого возможностями;
- разработку параллельных схем вычисления выражений;
- обеспечение эффективной «длинной арифметики» (для полного графа уже порядка 56 значение c_{12} не помещается в знаковый целый тип размером 64 бит);
- тестирование бинарных реализаций формул на кластере КарНЦ [29].

2. Формулы для подсчета коротких циклов в графе

В работе рассматриваются явные формулы, выражающие количество простых циклов длиной k в произвольных (c_k) и двудольных (c_{kb}) графах через матрицу смежности [13]. Имеются выражения для c_k при $k \leqslant 12$ и для c_{kb} при $k \leqslant 14$. Аддитивная структура формул позволяет спроектировать параллельную схему вычислений по ним. Однако для практической реализации такой схемы (на языках C и C++) потребовалось разработать процедуру трансляции выражений из системы компьютерной алгебры, поскольку встроенные средства системы не вполне справились с задачей. Кроме этого возник вопрос обеспечения достаточной разрядности вычислений, так как само количество циклов может не уместиться в стандартные типы целых чисел (до 64 бит).

Из всего набора формул наибольший интерес представляют выражения для величин c_{10}, c_{11}, c_{12}, c_{12b} и c_{14b}. При меньших значениях длины цикла достаточно эффективны обычные реализации формул в системе компьютерной алгебры на ПК. Выражения для c_8 и c_{10b} требуют нескольких часов счета при количестве вершин графа до 250, а остальные формулы справляются за это время с графами порядком 2000 и более.

2.1. Структура формул

Количество простых циклов длиной k выражается по принципу включения — исключения через числа замкнутых маршрутов длиной k, не являющихся простыми циклами. В результате упрощения формула принимает вид алгебраической суммы:

$$c_{k(b)} = \frac{1}{2k}\left(\operatorname{tr} A^k + a_2 t_2 + \ldots + a_s t_s\right), \tag{1}$$

где A — матрица смежности графа, a_i — целое число, а t_i — выражение количества маршрутов определенного вида через матрицу смежности. Для двудольных графов некоторые выражения t_i тождественно обращаются в нуль.

Любое выражение t_i представимо в виде суммы или кратной суммы, члены которой есть произведения определенных элементов матрицы смежности $A = (a_{ij})$ и ее степеней $A^k = (a_{ij}^{(k)})$. Например, для количества циклов длиной 6 получается следующая формула.

$$c_6 = \frac{1}{12}\left(\sum_{i=1}^{n} a_{ii}^{(6)} - 3\sum_{i=1}^{n}\left(a_{ii}^{(3)}\right)^2 - 6\sum_{i=1}^{n} a_{ii}^{(4)} d_i + 9\sum_{i=1}^{n}\sum_{j=1}^{n}\left(a_{ij}^{(2)}\right)^2 a_{ij} + \right.$$
$$\left. + 6\sum_{i=1}^{n} a_{ii}^{(4)} + 3\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^{(3)} + 4\sum_{i=1}^{n} d_i^3 - 4\sum_{i=1}^{n} a_{ii}^{(3)} - 12\sum_{i=1}^{n} d_i^2 + 4\sum_{i=1}^{n} d_i\right). \quad (2)$$

Величины d_i есть степени вершин графа и равны элементам $a_{ii}^{(2)}$. При значениях длины цикла не более 7 для произвольных графов и 8 для двудольных графов кратность всех сумм не превышает 2. При бóльших значениях появляются суммы кратности 3 и более.

Наряду с записями вида (2) в работе рассматриваются выражения с однократным использованием суммирования по каждому индексу:

$$c_{k(b)} = \frac{1}{2k}\sum_{i_1=1}^{n}\left(u + \ldots + \sum_{i_2=1}^{n}\left(v + \ldots + \left(\ldots + \sum_{i_m=1}^{n}(w + \ldots)\right)\right)\right), \quad (3)$$

где m — наибольшая кратность сумм, а $u, v, w, \ldots$ — члены различных сумм.

Таблица 1 содержит параметры представленного набора формул.

Таблица 1. Наибольшая кратность m сумм, количество s слагаемых, границы коэффициентов a_i и размеры l (кб) файлов, содержащих запись формул на языке системы Maple, для количества $c_{k(b)}$ циклов длиной 8–14 в произвольных и двудольных графах.

	c_{8b}	c_8	c_9	c_{10b}	c_{10}	c_{11}	c_{12b}	c_{12}	c_{14b}
m	2	4	4	4	5	5	6	6	6
s	20	35	58	59	160	341	230	958	1002
$\min a_i$	−132	−132	−1746	−2400	−5460	−68464	−72444	−389664	−3198384
$\max a_i$	73	272	1148	1680	10920	38016	54000	600588	2338560
l	0,4	0,9	2,1	1,7	6,2	16	9,2	51	52

2.2. Кодирование формул

В системе компьютерной алгебры Maple рассмотренные формулы для подсчета циклов (выведенные в этой системе) представляются естественным образом (2) или (3). Однако выполнение численных расчетов по ним в самой системе на ПК уже при длине цикла 10 и порядке графа 100 длится примерно сутки (без учета двудольности). Кроме того, не удается в желаемой мере перевести такие выражения на другие языки (в первую очередь, C) встроенными средствами. По причине огромных размеров формул (см. таблицу 1) неприемлемо и полностью ручное их перепрограммирование.

Бóльшую часть работы по переводу на язык C выполняют функции системы Maple 12 `codegen[prep2trans]` и `CodeGeneration[C]`. Собственно перевод осуществляет вторая процедура, однако она не рассчитана на функции `Sum`, `sum` и `add`, которые используются для представления сумм. Процедура `codegen[prep2trans]` выполняет предварительное преобразование этих функций в последовательности инструкций `for`. Применение указанных процедур Maple имеет ряд неудобств.

- Возведение в степень неизбежно транслируется в вызов функции `pow` и сопутствующие преобразования `int`/`double`/`int`.
- В качестве размеров массивов не принимаются идентификаторы.
- Нет возможности указать имя типа, отличное от стандартных (в частности, для типа «длинных чисел»).
- Слагаемые в сумме (еще до перевода) располагаются в произвольном порядке (как элементы структур наподобие множества). Для громоздких формул это существенно осложняет анализ кода и поиск фрагментов для ручной оптимизации.

Во избежание перечисленных недостатков была создана специальная процедура Maple взамен универсальных встроенных средств, ориентированная на трансляцию имеющихся формул. Функция работает с записями вида (2) и (3), а также со списками сумм, входящих в эти выражения. В таблице 2 представлены размеры C-файлов, сгенерированных этой процедурой.

Таблица 2. Размеры (кб) файлов с исходным кодом на языке C для формул $c_{k(b)}$ вида (2) и (3).

	c_{8b}	c_8	c_9	c_{10b}	c_{10}	c_{11}	c_{12b}	c_{12}	c_{14b}
(2)	2,1	3,9	7,4	7,2	22	54	34	164	177
(3)	1,1	1,9	3,5	3,4	9,8	24	15	72	78

2.3. Разрядность вычислений

Количество простых циклов фиксированной длины с ростом порядка графа может неограниченно увеличиваться. Например, для полного графа K_n значение c_k выражается соотношением [10, с. 60]

$$c_k = \frac{n!}{2\,k\,(n-k)!}. \tag{4}$$

Таков и результат вычислений по формулам (1), причем промежуточные значения могут оказаться еще больше, так как встречаются и положительные, и отрицательные величины.

Каждую сумму, входящую в выражение вида (2), можно записать, увеличив в общем случае ее кратность до некоторого m, так, что члены суммы будут составлены только из элементов матрицы смежности. Тогда значение суммы оценивается сверху величиной n^m, если заменить каждый член единицей, или точнее $n\cdot(n-1)^{m-1}$ с учетом того, что диагональные элементы матрицы смежности равны нулю, а все индексы суммирования располагаются в цепочку так, что каждый из них, кроме первого, встречается в паре хотя бы с одним из предшествующих.

Границы всех промежуточных результатов при вычислении выражений вида (2) или (3) наиболее просто оценить в худшем случае — когда положительные и отрицательные значения накапливаются отдельно. Отбросив в формуле для $c_{k(b)}$ делитель $2\,k$ (предполагается,

что деление выполняется в самом конце вычислений), разбив ее на две части по знаку слагаемых и оценив каждую сумму указанным выше способом, получим верхнюю ($\overline{c}_{k(b)}$) и нижнюю ($\underline{c}_{k(b)}$) оценки встречающихся в расчетах величин. Например, для (2)

$$\begin{aligned}\overline{c}_6 &= n\,(n-1)^5 + (9+6+3+4)\,n\,(n-1)^3 + 4\,n\,(n-1) = \\ &= n\,(n-1)\,(n^4 - 5\,n^3 + 28\,n^2 - 48\,n + 27), \qquad (5)\\ \underline{c}_6 &= -(3+6)\,n\,(n-1)^4 - (4+12)\,n\,(n-1)^2 = \\ &= -n\,(n-1)^2\,(9\,n^2 - 18\,n + 25). \qquad (6)\end{aligned}$$

Из полученных оценок приближенно находятся порядки графов, для которых вычисления при самой неудачной организации не превосходят заданную разрядность. В таблице 3 приведены результаты для знаковых целых типов размером 32, 64 и 128 бит.

Таблица 3. Нижние оценки наибольших порядков графов, вычисления для которых по формулам $c_{k(b)}$ не приводят к переполнению знаковых целых типов размером 32, 64 и 128 бит.

	c_{8b}	c_8	c_9	c_{10b}	c_{10}	c_{11}	c_{12b}	c_{12}	c_{14b}
32 бит	15	14	10	8	7	5	4	3	3
64 бит	235	235	128	79	79	52	38	36	21
128 бит	60097	60097	17696	6654	6654	2989	1535	1535	538

Практически допустимое количество вершин графа оказывается больше по причине неточности оценок и зависит от порядка вычисления сумм и от структуры графа. Размер 128 бит гарантирует достаточный запас для предполагаемых рабочих порядков графов при использовании формул $c_{k(b)}$.

3. Распараллеливание формул

Вложенные суммы вида (2) или (3) реализуются в программе на C++ с помощью вложенных циклов `for`, однако распараллеливание таких циклов для исполнения в нескольких процессах сопряжено с некоторыми трудностями, о которых идет речь в этом разделе.

Введем некоторые обозначения: `numOfProcs` — количество процессов, `id` — номер процесса, в котором исполняется код.

Пусть необходимо посчитать сумму вида

$$s = \sum_{i=1}^{n} a_i.$$

Такая сумма может быть посчитана с помощью одного цикла:

```
for ( i = 1; i <= n; i ++ )
    s = s + a [ i ];
```

В параллельном исполнении цикл выглядит следующим образом:

```
for ( i = id; i <= n; i += numOfProcs )
    local_s = local_s + a [ i ];
```

После выполнения цикла необходимо просуммировать локальные переменные `local_s` всех участвующих в суммировании процессов. В случае, когда `n` $<$ `numOfProcs`, `numOfProcs` $-$ `n` процессов будут простаивать.

Теперь рассмотрим r-кратную сумму вида

$$s = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \dots \sum_{i_r=1}^{n} a_{i_1,i_2,\dots,i_r}.$$

Для ее вычисления можно использовать систему вложенных циклов (именно в таком виде программа экспортируется из системы Maple):

```
for ( i1 = 1; i1 <= n; i1 ++ )
    for ( i2 = 1; i2 <= n; i2 ++ )
        ...
            for ( ir = 1; ir <= n; ir ++ )
                local_s = local_s + a [ i1 ] [ i2 ] ··· [ ir ];
```

Очевидная параллельная программа получается, когда распараллеливается внешний цикл:

```
for ( i1 = id; i1 <= n; i1 += numOfProcs )
    for ( i2 = 1; i2 <= n; i2 ++ )
        ...
            for ( ir = 1; ir <= n; ir ++ )
                local_s = local_s + a [ i1 ] [ i2 ] ··· [ ir ];
```

Но в данной реализации есть, по крайней мере, два недостатка. Первый состоит в том, что если $\mathtt{n} < \mathtt{numOfProcs}$, то часть процессов будет простаивать, и необходимо каким-либо образом распределить их на распараллеливание внутренних циклов. Второй недостаток заключается в том, что при $\mathtt{n} \equiv 1 \pmod{\mathtt{numOfProcs}}$, в частности при $\mathtt{n} = \mathtt{numOfProcs} + 1$, все процессы, кроме первого, будут выполнять одну итерацию внешнего цикла, а первый — две, то есть время работы всей программы для указанного значения n увеличится почти вдвое по сравнению со значением $\mathtt{n} = \mathtt{numOfProcs}$. При дальнейшем увеличении значения n время работы программы будет увеличиваться плавно, а следующий резкий скачок будет наблюдаться при $\mathtt{n} = 2 \cdot \mathtt{numOfProcs} + 1$. Такие значения n будем называть критическими.

Чтобы избежать указанных проблем, объединим два внешних цикла в один:

```
for ( i = id; i <= n * n; i += numOfProcs ) {
    i1 = ( i - 1 ) / n + 1;
    i2 = ( i - 1 ) % n + 1;
    for ( i3 = 1; i3 <= n; i3 ++ )
        ...
            for ( ir = 1; ir <= n; ir ++ )
                local_s = local_s + a [ i1 ] [ i2 ] ··· [ ir ];
}
```

При такой реализации одна итерация внешнего цикла имеет сложность в n раз меньшую, чем в первом случае, но количество самих итераций в n раз больше. В случае, когда $\mathtt{n}^2 \equiv 1 \pmod{\mathtt{numOfProcs}}$, все процессы кроме первого будут простаивать, когда первый выполняет последнюю итерацию. Но поскольку сама итерация исполняется в n раз быстрее, такая задержка оказывается несущественной (о чем свидетельствуют результаты вычислительного эксперимента). При такой реализации удается достигнуть линейного по количеству процессоров ускорения для любого значения n и любого количества вычислительных ядер $\mathtt{numOfProcs} \leqslant \mathtt{n}^2$.

Реализация, при которой объединяются три и более внешних циклов, оказывается менее эффективной на практике по причине того, что дальнейшего сглаживания времени работы при переходе значения n через критические значения не происходит, а скорость работы программы уменьшается из-за более громоздкого вычисления индексов i1, i2, i3, ...

4. Реализация формул

4.1. Вычислительная среда

Кластер КарНЦ [29], на котором проводились вычислительные эксперименты, состоит из 10 вычислительных узлов. Каждый узел содержит два четырехъядерных процессора Quad-Core Intel Xeon 5430 с частотой одного ядра 2,66 ГГц и оперативную память объемом 4 Гб. Операционная система: SUSE Linux Enterprise Server 10. Компилятор: Intel C++ 10.1.

В силу ограничений на использование кластера, эксперименты проводились на 1, 2, 4, 8, 16, 32 и 64-х ядрах.

4.2. Варианты реализации

Для того чтобы оптимизировать какую-либо реализацию формул, необходимо проверить различные их варианты. В первую очередь, отметим, что реализации могут отличаться способом вычисления: формула вида (2) экспортируется в язык C++ в виде последовательности не связанных друг с другом групп вложенных циклов, каждая из которых вычисляет одно слагаемое, в то время как формула вида (3) экспортируется в виде одной группы вложенных циклов, а слагаемые формулы записываются подряд на своем уровне вложенности. Во-вторых, у каждого слагаемого есть постоянный множитель, который можно вынести за знак суммы. Таким образом, получаются 4 варианта реализации, из которых необходимо выбрать те, которые работают быстрее (если это возможно).

Поскольку все формулы однотипны и представляют собой последовательности сумм, указанный выбор можно сделать, проведя короткий эксперимент над формулами для некоторых графов разного порядка.

В первую очередь, были протестированы непосредственные реализации формул вида (2) и вида (3), получаемые в результате экспортирования соответствующих выражений из системы Maple. Тестирование проводилось для циклов длиной 12 в полном графе из n вершин ($n = 12, 13, \ldots, 50$). Суммарное время работы показало, что реализация вида (3) оказалась быстрее реализации вида (2) в 10 раз. Объяснить такое различие можно тем, что компилятор лучше оптимизирует объединенные циклы, чем разделенные. Таким образом, далее в работе рассматриваются реализации вида (3), в которой циклы объединены.

Реализация формулы, в которой постоянные множители вынесены за знак суммы, оказалась быстрее реализации с множителями внутри суммы приблизительно на 7%. Поэтому именно такая реализация была взята за основу параллельной программы и именно с этой реализацией проводилась «ручная» оптимизация, описанная ниже в разделе 4.4.

4.3. «Длинная арифметика»

Как было показано в разделе 2.3, для того чтобы находить количество циклов в графах с сотнями вершин, требуется 128-битовая арифметика. На языке C++ такая арифметика реализована в виде класса `Int128` с определенными для него операторами сложения, умножения и деления на стандартное 32-битовое целое. Переменные такой разрядности способны хранить величины приблизительно до $1{,}7 \cdot 10^{38}$ по абсолютному значению.

Используемая в работе реализация класса `Int128` обладает той особенностью, что она выполнена без использования языка ассемблера. Это позволяет компилировать программу любым компилятором языка C++ без необходимости переписывать часть кода. Однако недостатком такой реализации явилось то обстоятельство, что при замене стандартного 64-битового типа данных языка C++ на класс `Int128` работа программы замедлилась приблизительно в 28 раз. По этой причине формулы нуждались в дополнительной ручной оптимизации.

4.4. Ручная оптимизация формул

Длинная арифметика, существенно замедляющая исполнение программы, требуется не для всех операций в процессе вычислений. Для примера рассмотрим одну из самых сложных частей формулы для количества циклов длиной 12:

$$186 \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \sum_{i_3=1}^{n} \sum_{i_4=1}^{n} \sum_{i_5=1}^{n} \sum_{i_6=1}^{n} a_{i_1 i_6} a_{i_1 i_2} a_{i_1 i_3} a_{i_4 i_6} a_{i_2 i_5} a_{i_5 i_6} a_{i_3 i_5} a_{i_3 i_6} a_{i_1 i_4} a_{i_2 i_3} a_{i_2 i_4} a_{i_4 i_5}.$$

В этой сумме встречаются только элементы матрицы смежности, которые равны 0 или 1, поэтому их перемножение можно выполнять в стандартном 8-битовом целочисленном типе данных, а суммирование — в 64-битовом, заменив операцию умножения операцией побитового «И». Всего имеется 6 шестикратных сумм в формуле для подсчета циклов длиной 12. В пятикратных суммах встречаются элементы квадрата и куба матрицы смежности (верхняя оценка на максимальный элемент у куба матрицы смежности есть n^2), поэтому умножение можно выполнять в стандартном 32-битовом целочисленном типе данных.

Такая оптимизация в шестикратной и пятикратной суммах уменьшила время вычисления по формуле в 15,6 раза. Ручная оптимизация в четырехкратной сумме, во-первых, крайне затруднительна по причине того, что в ней 240 слагаемых, а во-вторых, сомнительна, так как эти слагаемые содержат произведения элементов 5-х степеней матрицы смежности (верхняя оценка на максимальный элемент 5-й степени матрицы смежности есть n^4) и само перемножение необходимо проводить в 64-битовом типе данных, а суммировать — в 128-битовом. Кроме того, как показал эксперимент, четырехкратные суммы (а также суммы меньшей кратности) занимают меньше 1% времени вычислений при значениях n, близких к 100.

5. Результаты экспериментов

Для тестирования были выбраны классы полных и полных двудольных графов, поскольку для этих классов легко проверить правильность ответа программы и вычисления для них наиболее длительны (в других случаях срабатывают оптимизации, при которых запрещается перемножение и суммирование нулевых элементов матрицы смежности), что дает верхнюю оценку времени работы программы на всех остальных графах.

На рис. 1 приводится зависимость времени работы программы для подсчета количества циклов длиной 11 и 12 в полном графе от количества вершин. По графикам видно, что время работы программы монотонно увеличивается с ростом порядка графа, что свидетельствует о равномерной загрузке процессоров для любого значения n. Расстояние между графиками в логарифмических координатах остается одинаковым при удвоении количества вычислительных ядер. Это означает, что достигается линейное по числу ядер ускорение. В целом, изломы на графиках отсутствуют, что означает отсутствие скачков времени работы программы при переходе числа вершин через критическую область (когда число вершин $\mathtt{n} \equiv \mathtt{1} \pmod{\mathtt{numOfProcs}}$).

На рис. 2 сравнивается время вычислений по формулам для произвольных и двудольных графов. Важно отметить, что время подсчета циклов длиной 14 в полном двудольном графе меньше, чем время подсчета циклов длиной 12 в полном графе.

В таблице 4 приводятся максимальные порядки графов, для которых время вычисления по явным формулам не превышает одного часа.

Более подробные результаты тестирования с указанием точного времени работы программ можно отыскать на сайте [30].

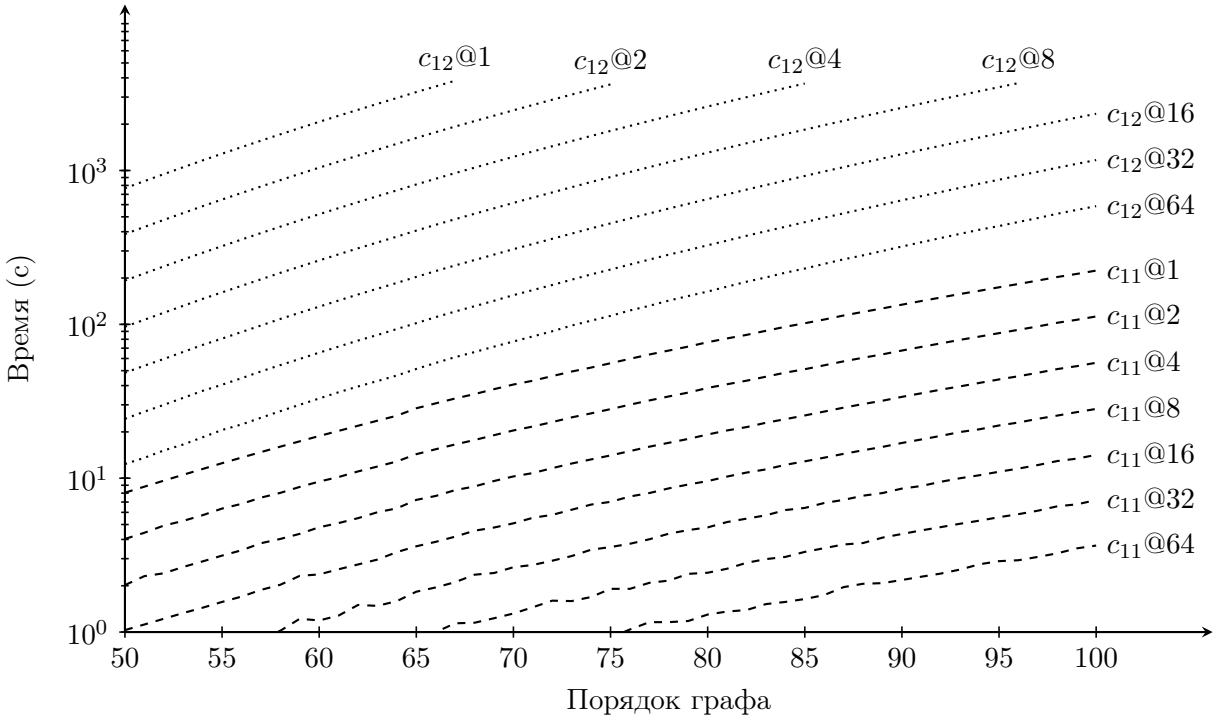

Рис. 1. Подсчет циклов длиной 11 и 12 в произвольном графе по явным формулам при разном количестве процессоров на кластере КарНЦ [29].

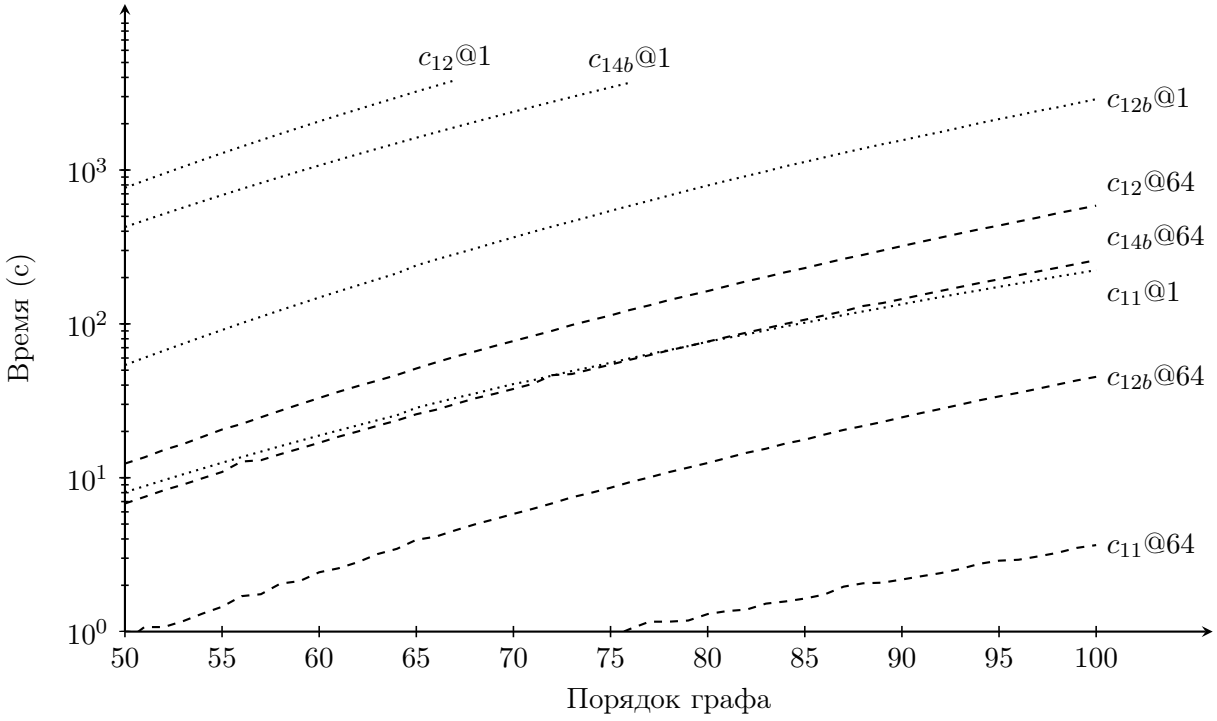

Рис. 2. Подсчет циклов длиной 11 и 12 в произвольном графе и длиной 12 и 14 в двудольном графе по явным формулам на одном и 64-х процессорах кластера КарНЦ [29].

6. Обсуждение результатов

Экстраполируя результаты эксперимента, частично приведенные на рис. 1 и 2 и в таблице 4, можно сделать вывод о времени работы программ на большем количестве ядер.

Таблица 4. Наибольшие порядки графов, при которых вычисления по формулам $c_{k(b)}$ на кластере КарНЦ [29] длятся не более одного часа.

$c_{k(b)}$	Количество ядер						
	1	2	4	8	16	32	64
c_{11}	175	202	232	267	307	353	403
c_{12}	66	74	84	95	107	121	136
c_{12b}	104	117	131	148	166	187	206
c_{14b}	75	86	97	110	124	143	162

Например, если известно, что на 64 ядрах число циклов длиной 12 в графе из 100 вершин вычисляется за 586,6 секунды, то на 640 ядрах время вычисления составит приблизительно 59 секунд, а максимальный порядок графа, для которого вычисления займут не больше одного часа, — приблизительно 190.

Учет специфики графа позволяет оптимизировать формулы для конкретного случая и достигать таким образом значительного ускорения. Один из примеров был представлен выше: учет двудольности графа позволяет подсчитывать циклы длиной 14 в двудольных графах быстрее, чем циклы длиной 12 в произвольных графах.

Другой пример связан с циклами длиной 9 в графе шахматного ферзя на доске $n \times n$. При использовании общей формулы для c_9 за один час на 64 ядрах можно подсчитать циклы при $n = 46$ (2116 вершин). По сравнению с матрицей смежности полного графа матрица смежности ферзевого графа содержит достаточно много нулей, поэтому хорошей оптимизацией в данном случае будет добавление в программу дополнительных условий, предотвращающих перемножение и суммирование заведомо нулевых элементов, а также избавление от 128-битной арифметики в большем количестве слагаемых, чем для полного графа. После такой оптимизации за один час на 64 ядрах можно проводить вычисления при $n = 120$ (14400 вершин), а при $n = 46$ программа стала работать всего 22 секунды.

Литература

1. Harary F., Ross I.C. The Number of Complete Cycles in a Communication Network // The Journal of Social Psychology. — November 1954. — Vol. 40, Part 2. — P. 329–332.

2. Bianconi G., Capocci A. Number of Loops of Size h in Growing Scale-Free Networks // Physical Review Letters. — February 2003. — Vol. 90, N. 7. — P. 078701(4).

3. Cardillo A., Scellato S., Latora V., Porta S. Structural Properties of Planar Graphs of Urban Street Patterns // Physical Review E. — June 2006. — Vol. 73, Issue 6. — P. 066107(8).

4. Fagiolo G. Clustering in Complex Directed Networks // Physical Review E. — August 2007. — Vol. 76, Issue 2. — P. 026107(8).

5. Halford T.R., Chugg K.M. An Algorithm for Counting Short Cycles in Bipartite Graphs // IEEE Transactions on Information Theory. — January 2006. — Vol. 52, N. 1. — P. 287–292.

6. Харари Ф. Теория графов. — М. : Мир, 1973. 300 с.

7. Харари Ф., Палмер Э. Перечисление графов. — М. : Мир, 1977. 324 с.

8. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. — М. : Мир, 1982. 416 с.

9. Flum J., Grohe M. The Parameterized Complexity of Counting Problems // SIAM Journal on Computing. — May 2004. — Vol. 33, N. 4. — P. 892–922.

10. Harary F., Manvel B. On the Number of Cycles in a Graph // Matematický časopis. — 1971. — Vol. 21, N. 1. — P. 55–63.

11. Alon N., Yuster R., Zwick U. Finding and Counting Given Length Cycles // Algorithmica. — March 1997. — Vol. 17, N. 3. — P. 209–223.

12. Chang Y.C., Fu H.L. The Number of 6-Cycles in a Graph // The Bulletin of the Institute of Combinatorics and Its Applications. — 2003. — Vol. 39. — P. 27–30.

13. Perepechko S.N., Voropaev A.N. The Number of Fixed Length Cycles in an Undirected Graph. Explicit Formulae in Case of Small Lengths // Mathematical Modeling and Computational Physics (MMCP'2009): Book of Abstracts of the International Conference (Dubna, July 7–11, 2009). — Dubna : JINR, 2009. — P. 148–149.

14. Хоменко Н.П., Головко Л.Д. Выделение из графа его частей некоторых типов и подсчет их количества // Украинский математический журнал. — Май–июнь 1972. — Т. 24, №3. — С. 385–396.

15. Хоменко Н.П., Шевченко Е.Н. К проблеме выделения и подсчета // Украинский математический журнал. — Март–апрель 1978. — Т. 30, №2. — С. 201–211.

16. Welch J.T. Jr. A Mechanical Analysis of the Cyclic Structure of Undirected Linear Graphs // Journal of the ACM. — April 1966. — Vol. 13, N. 2. — P. 205–210.

17. Tiernan J.C. An Efficient Search Algorithm to Find the Elementary Circuits of a Graph // Communications of the ACM. — December 1970. — Vol. 13, N. 12. — P. 722–726.

18. Mateti P., Deo N. On Algorithms for Finding All Circuits of a Graph. — Urbana, 1973. — 36 p. (UIUCDCS-R-73-585, Department of Computer Seience, University of Illinois, Illinois).

19. Tarjan R. Enumeration of the Elementary Circuits of a Directed Graph // SIAM Journal on Computing. — September 1973. — Vol. 2, N. 3. — P. 211–216.

20. Johnson D.B. Finding All the Elementary Circuits of a Directed Graph // SIAM Journal on Computing. — March 1975. — Vol. 4, N. 1. — P. 77–84.

21. Gibbs N.E. Algorithm 492: Generation of All the Cycles of a Graph from a Set of Basic Cycles // Communications of the ACM. — June 1975. — Vol. 18, N. 6. — p. 310.

22. Szwarcfiter J.L., Lauer P.E. A Search Strategy for the Elementary Cycles of a Directed Graph // BIT Numerical Mathematics. — June 1976. — Vol. 16, N. 2. — P. 192–204.

23. Sysło M.M. An Efficient Cycle Vector Space Algorithm for Listing All Cycles of a Planar Graph // SIAM Journal on Computing. — November 1981. — Vol. 10, N. 4. — P. 797–808.

24. Liu H., Wang J. A New Way to Enumerate Cycles in Graph // Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), February 19–25, 2006, Guadeloupe, French Caribbean, Proceedings. — 2006. — p. 57.

25. Bax E.T. Inclusion and Exclusion Algorithm for the Hamiltonian Path Problem // Information Processing Letters. — September 1993. — Vol. 47, Issue 4. — P. 203–207.

26. Bax E.T. Algorithms to Count Paths and Cycles // Information Processing Letters. — December 1994. — Vol. 52, Issue 5. — P. 249–252.

27. Лихтенбаум Л.М. О гамильтоновых циклах связного графа // Кибернетика. — 1968. — №4. — С. 72–75.

28. Doğrusöz U., Krishnamoorthy M.S. Enumerating All Cycles of a Planar Graph // Parallel Algorithms and Applications. — 1996. — Vol. 10, N. 1–2. — P. 21–36.

29. Центр высокопроизводительной обработки данных ЦКП КарНЦ РАН [электронный ресурс] / Институт прикладных математических исследований Карельского научного центра РАН © 2009. — URL: `http://cluster.krc.karelia.ru`. — Загл. с экрана.

30. FlowProblem [электронный ресурс] / Copyright © 2008–2009 — Artem M. Karavaev. — URL: `http://www.flowproblem.ru`. — Загл. с экрана. — Яз. англ.