УДК 004.42, 519.1 doi: 10.15827/0236-235X.142.374-383 2024. Т. 37. № 3. С. 374–383

Реберная раскраска кубического графа в задаче распараллеливания расчетов на неструктурированной поверхностной расчетной сетке

А.А. Гуличева [1], А.А. Рыбаков [2, 3]✉

[1] МИРЭА – Российский технологический университет, г. Москва, 119454, Россия
[2] Национальный исследовательский центр «Курчатовский институт», г. Москва, 123182, Россия
[3] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, г. Москва, 119334, Россия

Ссылка для цитирования
Гуличева А.А., Рыбаков А.А. Реберная раскраска кубического графа в задаче распараллеливания расчетов на неструктурированной поверхностной расчетной сетке // Программные продукты и системы. 2024. Т. 37. № 3. С. 374–383. doi: 10.15827/0236-235X.142.374-383
Информация о статье
Группа специальностей ВАК: 2.3.5
Поступила в редакцию: 06.05.2024 После доработки: 10.06.2024 Принята к публикации: 14.06.2024

Аннотация. В статье рассмотрен вопрос повышения производительности работы конечно-объемных численных методов на вычислительных системах с общей памятью. В этих методах на этапе расчета перетекания потоков консервативных величин через границы расчетных ячеек возможны конфликты по данным, что приводит к снижению производительности, особенно при большом количестве одновременно работающих потоков. Для устранения конфликтов по данным предлагается решение, основанное на разбиении множества обрабатываемых границ ячеек расчетной сетки на подмножества без конфликтов и на обработке этих множеств по отдельности. Решение рассматривается на примере расчетов на поверхностных неструктурированных расчетных сетках, для которых поставленная задача сводится к задаче построения реберной раскраски кубического графа. Для построения реберной раскраски кубического графа применены два алгоритма: тривиальный линейный алгоритм раскраски в пять цветов и алгоритм построения раскраски Тейта в три цвета. Проводится сравнение двух алгоритмов раскраски, а также замеряется влияние предложенного подхода на эффективность распараллеливания расчетов на поверхностной расчетной сетке. Использование подхода избавления от зависимостей по данным с помощью реберной раскраски кубического графа протестировано на численной задаче расчета обледенения поверхности тела. Запуски выполнялись на микропроцессоре Intel Xeon Phi с большим количеством параллельных потоков. Результаты показали, что при возрастании количества потоков до 144 и более эффективность распараллеливания при использовании реберной раскраски вдвое выше, чем при обычном подходе по устранению зависимостей с помощью директив OpenMP.

Ключевые слова: численные методы, распараллеливание, векторизация, расчетная сетка, дуальный граф, реберная раскраска, раскраска Тейта

Благодарности. Работа выполнена в рамках госзадания НИЦ «Курчатовский институт» по теме FNEF 2024 0016

Введение. При изучении природных явлений и проектировании сложных технических систем используется компьютерное моделирование физических процессов, описываемых в виде систем дифференциальных уравнений в частных производных. Одним из наиболее распространенных видов численных методов, которые применяются в компьютерном моделировании для решения таких систем уравнений, является метод конечных объемов. Конечно-объемные численные методы активно используются при численном решении задач газовой динамики, теории мелкой воды и многих других [1–3]. При этом постоянно ведутся исследования, направленные на максимальное использование возможностей массивно-параллельных вычислительных систем для решения таких задач [4–6]. Отдельно следует отметить исследования, направленные на максимальное использование возможностей векторизации решателей различного вида [7]. Векторизация вычислений позволяет кратно увеличивать производительность программного кода путем объединения однотипных скалярных операций в векторные инструкции. Современные микропроцессоры Intel поддерживают набор векторных инструкций AVX-512, с помощью которых можно векторизовать не только безусловные операции, но и программный код, содержащий разветвленное управление, команды переходов и вызовы функций. Для успешной векторизации программного кода важно избавиться от зависимостей между отдельными векторизуемыми элементами (например, при векторизации цикла следует минимизировать зависимости между его итерациями). В работе [8] рассматривается векторизация трехмерного газодинамического решателя, и можно заметить, что наименьшая эффективность векторизации наблюдается для функции пересчета потоков консервативных величин между расчетными ячейками.

Данная работа посвящена задаче, решаемой на неструктурированной поверхностной расчетной сетке, – моделированию обледенения поверхности обтекаемого тела [9].

Моделирование процесса обледенения поверхности тела выполняется итерационно. На каждой итерации расчетов в каждой расчетной ячейке решается система уравнений массового и теплового балансов, из которой получаются основные данные состояния расчетной ячейки (температура, количество жидкости и накопленного льда). Между расчетными итерациями выполняется моделирование протекания потоков жидкости в соседних ячейках (массы и тепла через границы ячеек) [10–12]. Также в зависимости от настроек решателя через заданные промежутки времени выполняется пересчет геометрии поверхности тела за счет накопленной в ее ячейках массы льда. Будем считать, что расчеты обледенения выполняются на неструктурированной поверхностной расчетной сетке с треугольными ячейками. Пересчет состояния ячейки происходит независимо от других ячеек, данные вычисления могут производиться параллельно. Пересчет потоков массы и тепла осуществляется для каждого ребра домена расчетной сетки, при этом поток перетекает из одной инцидентной ребру ячейки в другую.

Общая схема пересчета потоков для одного ребра может выглядеть так:

```
void
Edge::calc_flows()
{
    // Получение пары ячеек.
    Cell* fst = ...;
    Cell* sec = ...;

    // Вычисление потоков.
    double flow_w = ...;
    double flow_q = ...;

    // Корректировка потоков.
    fst->flow_out_w += flow_w;
    fst->flow_out_q += flow_q;
    sec->flow_out_w -= flow_w;
    sec->flow_out_q -= flow_q;
}
```

Для выполнения пересчета потоков для всех ребер домена расчетной сетки необходимо обработать все ребра в цикле:

```
for (auto e : own_edges)
{
    e->calc_flows();
}
```

При обработке ребер домена расчетной сетки в цикле возникает желание распараллелить или векторизовать данный цикл. При распараллеливании вычислений с помощью OpenMP следует учитывать конфликты по данным, которые могут возникнуть при корректировке потоков (если несколько потоков начнут одновременно изменять значение одной области памяти). Для устранения этих конфликтов достаточно выполнять операции корректировки потоков в атомарном режиме:

```
#pragma omp atomic
fst->flow_out_w += flow_w;
#pragma omp atomic
fst->flow_out_q += flow_q;
#pragma omp atomic
sec->flow_out_w -= flow_w;
#pragma omp atomic
sec->flow_out_q -= flow_q;
```

Использование #pragma omp atomic гарантирует, что указанная команда одновременно будет обрабатываться только одним потоком (то есть между чтением старого значения переменной и записью нового значения не вклинится другой поток). При большом количестве используемых потоков это может приводить к потерям производительности. Отдельно следует отметить, что такой подход не сработает при попытке векторизации рассматриваемого цикла. Зачастую вычисление самих потоков через ребро не содержит сложного управления и может быть легко векторизовано, но векторизации всего цикла мешают зависимости по данным при корректировке потоков. Возникает желание разбить исходное множество ребер сетки на такие подмножества, чтобы параллельная обработка ребер каждого отдельного подмножества не приводила к возникновению конфликтов при корректировке потоков.

Сведение задачи распараллеливания вычислений к реберной раскраске

Будем решать задачу разрешения конфликтов между ребрами расчетной сетки с помощью реберной раскраски графа конфликтов. Такой подход является достаточно естественным [13], однако в рассматриваемом случае вершины графа конфликтов будут иметь небольшую степень, что свидетельствует о допустимости реберной раскраски в небольшое количество цветов.

Рассмотрим ситуацию, при которой возможно возникновение конфликта при корректировке потоков во время параллельной обработки двух ребер расчетной сетки. Такой конфликт возможен в том случае, когда оба обрабатыва-

емых ребра являются инцидентными одной и той же ячейке. Рассмотрим дуальный граф расчетной сетки – то есть граф, вершины которого соответствуют ячейкам расчетной сетки, а ребро проведено между двумя вершинами только в том случае, когда две соответствующие ячейки расчетной сетки являются соседними по ребру. Без ограничения общности будем считать, что рассматриваемая расчетная сетка не имеет краев, то есть каждая ее ячейка имеет ровно трех соседей. В этом случае ее дуальный граф будет кубическим. При этом задача разбиения ребер расчетной сетки на неконфликтующие множества сводится к построению реберной раскраски дуального графа. В процессе нахождения реберной раскраски построенного дуального графа возникает вопрос о том, в какое минимальное количество цветов можно раскрасить этот граф.

Сразу следует отметить, что ребра произвольного кубического графа можно тривиальным способом раскрасить в пять цветов, поскольку у любого ребра есть ровно четыре смежных ребра, которые должны отличаться от него по цвету, поэтому для него всегда можно выбрать пятый цвет. Данный алгоритм является линейным и интереса не представляет. С другой стороны, очевидно, что двумя цветами обойтись не удастся, так как в каждой вершине сходятся по три разноцветных ребра. Если кубический граф допускает правильную реберную раскраску в три цвета, то такая раскраска называется раскраской Тейта. Следует заметить, что не все кубические графы допускают раскраску Тейта. В работе [14] предлагается способ построения раскраски Тейта для плоских кубических графов. Авторы данной работы будут использовать его в качестве основы. Кубический граф, порожденный поверхностной расчетной сеткой, не всегда является плоским, но попытка поиска для него раскраски Тейта может быть оправданна. К тому же для целей практического использования не будет большой проблемой отсутствие раскраски Тейта, в этом случае всегда можно воспользоваться тривиальной раскраской в пять цветов.

Редуцирование кубического графа по ребру и его восстановление

Рассмотрим принцип и реализацию алгоритма построения раскраски Тейта для кубического графа, являющегося дуальным графом для замкнутой поверхностной неструктурированной расчетной сетки. В работах [14, 15] рассматривается подход к построению раскраски Тейта путем удаления ребер из исходного плоского кубического графа. Ребра удаляются до тех пор, пока не будет получен кубический граф с уже известной раскраской. После этого ребра возвращаются в граф в обратном порядке с соответствующей коррекцией раскраски. Операцию удаления ребра из графа будем называть редуцированием кубического графа по ребру. Чтобы редуцирование кубического графа по ребру можно было использовать для построения раскраски Тейта, необходимо уметь провести последовательность редукций до достижения простого по структуре кубического графа, раскраска которого не представляет сложности. При этом в получающихся графах допустимо наличие параллельных ребер, однако запрещено появление петель, так как раскраска Тейта для кубического графа с петлями невозможна. Опишем операцию редуцирования графа более подробно.

Сначала рассмотрим выполнение редуцирования кубического графа по ребру e с концами v_1 и v_2, где инцидентными ребрами вершины v_1 являются ребра e, $e_1(v_1)$, $e_2(v_1)$, а инцидентными ребрами вершины v_2 являются ребра e, $e_1(v_2)$, $e_2(v_2)$, а также среди ребер e, $e_1(v_1)$, $e_2(v_1)$, $e_1(v_2)$, $e_2(v_2)$ нет параллельных (рис. 1, слева). Из этого следует, что существует только одно ребро, соединяющее вершины v_1 и v_2. В этом случае будем говорить, что редуцирование выполняется по уникальному ребру e.

При выполнении редуцирования по ребру e само ребро e удаляется, также удаляются вершины v_1 и v_2, ребра $e_1(v_1)$ и $e_2(v_1)$ склеиваются в результирующее ребро re_1, ребра $e_1(v_2)$ и $e_2(v_2)$ склеиваются в ребро re_2 (рис. 1, справа).

Другим случаем редуцирования является вариант, при котором концами рассматриваемого ребра e также являются вершины v_1 и v_2, однако между ними проходит еще одно ребро, без ограничения общности будем считать, что это ребро $e_2(v_1) = e_2(v_2)$ (рис. 2, слева). В этом случае при редуцировании удаляются ребро e,

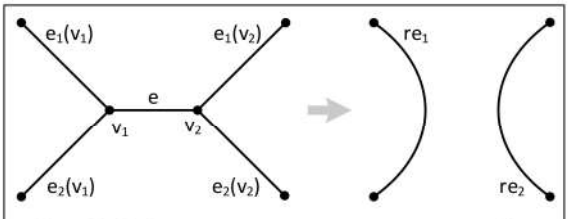

Рис. 1. Редуцирование по уникальному ребру

Fig. 1. Reduction by unique edge

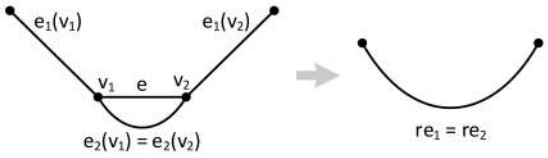

Рис. 2. Редуцирование по параллельному ребру

Fig. 2. Reduction by parallel edge

вершины v_1 и v_2, а все три ребра $e_1(v_1)$, $e_2(v_1) = e_2(v_2)$, $e_1(v_2)$ склеиваются в единое ребро $re_1 = re_2$ (рис. 2, справа). Такой шаг редуцирования будем называть редуцированием по параллельному ребру.

Отдельно отметим случай, когда между двумя вершинами проходят три параллельных ребра. Если мы достигли такого графа, то это и есть минимальный кубический граф, построение раскраски для которого очевидно и от которого нужно двигаться в обратную сторону, постепенно восстанавливая исходный граф. Также следует рассмотреть случай, при котором после редуцирования по уникальному ребру граф перестает быть связным. Это значит, что в исходном графе ребро e было мостом, такие графы рассматривать не будем. В других случаях в графе найдется либо уникальное, либо параллельное ребро, по которому можно осуществить следующий шаг редуцирования.

В качестве примера рассмотрим редуцирование графа, представляющего собой куб, как показано на рисунке 3.

Кубический граф содержит 8 вершин v_0–v_7 и 12 ребер e_0–e_{11}. Будем считать, что нижние индексы в именах вершин и ребер являются также их идентификаторами. Для полного редуцирования указанного графа требуется выполнить три шага, которые могут быть записаны в историю редуцирования следующим образом:

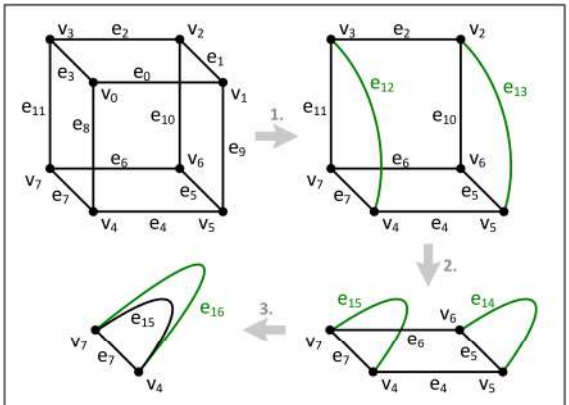

Рис. 3. Полное редуцирование графа

Fig. 3. Graph full reduce

```
   e0 [(v0 : e3, e8 -> re12),
(v1 : e1, e9 -> re13)]
   e2 [(v2 : e13, e10 -> re14),
(v3 : e12, e11 -> re15)]
   e5 [(v5 : e4, e14 -> re16),
(v6 : e6, e14 -> re16)]
```

По такой записи истории редуцирования можно идентифицировать каждый шаг, определить его тип (редуцирование по уникальному ребру или по параллельному) и восстановить граф.

Опираясь на указанные операции редуцирования, можно описать алгоритм построения раскраски Тейта во время восстановления графа.

Алгоритм реберной раскраски кубического графа в три цвета

Прежде чем перейти к описанию алгоритма построения раскраски, рассмотрим центральный объект, который будет использован в данном построении. Без ограничения общности будем считать, что выполняется раскраска в красный, синий и зеленый цвета. Эти же цвета используются и на иллюстрациях. Пусть ребра некоторого кубического графа правильным образом раскрашены в три цвета. Возьмем два произвольных цвета, например, красный и синий. Если рассмотреть все покрашенные в них ребра, а также все инцидентные им вершины, то получим граф порядка 2, у которого в каждой вершине сходятся разноцветные ребра. Очевидно, что такой граф является объединением простых циклов четной длины (двухцветных циклов). На рисунке 4 приведен пример такого двухцветного красно-синего цикла.

С таким двухцветным циклом можно выполнять следующие операции. Во-первых, для всех ребер двухцветного цикла можно заменить цвет на противоположный, после чего раскраска в исходном графе останется правильной (рис. 4). Также можно применить перекраску в противоположный цвет не всех ребер цикла, а только расположенных между двумя фиксированными ребрами e_1 и e_2 (на рисунке 5 приведены два варианта такой перекраски в порядке обхода цикла по часовой стрелке и против). При такой перекраске ребер в цикле возникают два конфликта по цветам: между ребрами e_1 и e_2 и их перекрашенными соседями.

Смысл частичной замены цветов в двухцветном цикле становится понятным, если возникает необходимость поместить в граф новое ребро. На рисунке 6 продемонстрирована операция, при которой выполняется частичная перекраска цикла между ребрами e_1 и e_2, а затем на ребра e_1 и e_2 добавляются новые вершины v_1 и v_2 соответственно, между которыми проводится ребро. Полученные после разбиения ребер e_1 и e_2 более мелкие ребра перекрашиваются для устранения конфликтов. Новое проведенное ребро при этом перекрашивается в третий, свободный, цвет, что приводит к сохранению правильной раскраски во всем графе. Таким образом, выбрав два произвольных ребра на любом двухцветном цикле, можно добавить новое ребро с концами на выбранных ребрах, а затем перекрасить ребра и сохранить правильную реберную раскраску. Такую операцию будем называть восстановлением ребра по двухцветному циклу.

Также понадобится операция поиска двухцветного цикла, начиная с произвольного ребра e, содержащего ребра цветов color(e) и $\text{color}_2 \neq \text{color}(e)$. Такой цикл всегда существует, и он ровно один.

Теперь, имея в своем распоряжении три простые операции: поиск двухцветного цикла по ребру и второму цвету, перекраска двухцветного цикла и восстановление ребра по двухцветному циклу, – можно описать алгоритм восстановления одного шага редуцирования графа с сохранением правильной реберной раскраски (рис. 7).

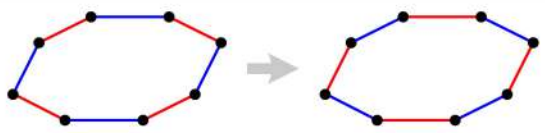

Рис. 4. Полная замена цветов в двухцветном цикле

Fig. 4. Full color switch in bicolor cycle

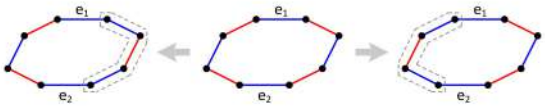

Рис. 5. Частичная замена цветов в двухцветном цикле

Fig. 5. Partial color switch in bicolor cycle

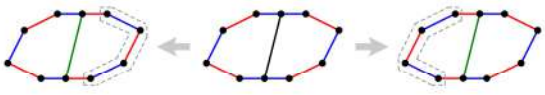

Рис. 6. Замена цветов в двухцветном цикле при добавлении нового ребра

Fig. 6. Color switch in bicolor cycle when adding a new edge

Дополнительно следует отметить, что восстановление шага редуцирования графа по параллельному ребру не представляет сложности, так как это локальная операция, которая затрагивает единственное результирующее ребро. Поэтому рассмотрим подробно только восстановление шага редуцирования по уникальному ребру.

Пусть имеется шаг редуцирования графа, результирующими ребрами после выполнения которого являются различные ребра re_1 и re_2. В процессе восстановления нужно поместить на эти ребра новые вершины v_1 и v_2 соответственно, провести между ними ребро и выполнить коррекцию раскраски. Опишем последовательность действий в данном случае.

Вариант 1. Если результирующие ребра re_1 и re_2 имеют разные цвета, следует найти двухцветный цикл, начиная с ребра re_1, включающий в себя ребра цветов color(re_1) и color(re_2). Если найденный цикл содержит также и ребро re_2, то можно восстановить исходное ребро по найденному двухцветному циклу. В противном случае найденный двухцветный цикл перекрашивается. Поскольку перекрашивание затронет только ребро re_1, после этой операции получится ситуация, в которой ребра re_1 и re_2 имеют один цвет. В этом случае осуществляется переход ко второму варианту.

Вариант 2. Если результирующие ребра re_1 и re_2 имеют один и тот же цвет, то рассмотрим два оставшихся цвета – x и y, а также два двухцветных цикла, начиная с ребра re_1. Причем в первый цикл будем включать ребра с цветами color(re_1) и x, а во второй ребра с цветами color(re_1) и y. Если один из найденных двухцветных циклов будет содержать также и ребро re_2, то по этому циклу и нужно выполнить восстановление исходного ребра, которое было удалено из графа при редуцировании. Отметим, что в общем случае нет гарантии, что один из найденных двухцветных циклов будет содержать ребро re_2 (исследования в работе [14] касаются только плоских кубических графов, а дуальный граф поверхностной сетки необязательно будет плоским), в этом случае корректировка реберной раскраски невозможна и исходных граф не допускает раскраски Тейта. Однако в ходе экспериментов по раскраске кубических дуальных графов, построенных по замкнутым поверхностным расчетным сеткам,

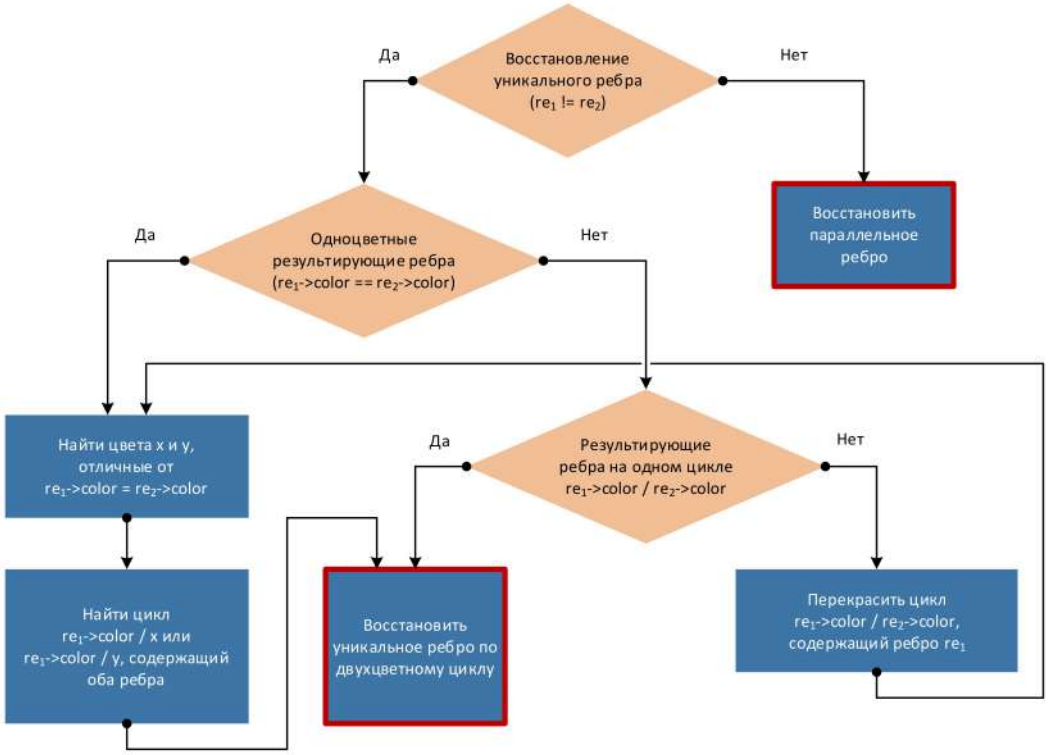

Рис. 7. Итерация алгоритма восстановления и перекраски графа

Fig. 7. Iteration of graph restore and repaint algorithm

такая ситуация не наблюдалась и для всех таких графов раскраска Тейта была найдена.

На рисунке 8 приведена последовательность восстановления графа, редуцирование которого продемонстрировано на рисунке 3. Приведем еще раз историю редуцирования графа:

```
  e0 [(v0 : e3, e8 -> re12),
(v1 : e1, e9 -> re13)]
  e2 [(v2 : e13, e10 -> re14),
(v3 : e12, e11 -> re15)]
  e5 [(v5 : e4, e14 -> re16),
(v6 : e6, e14 -> re16)]
```

Первый шаг восстановления относится к редуцированию по параллельному ребру. Ребро e_{16} разбивается на ребра e_6, e_{14}, e_4 с помощью вершин v_6 и v_5. Между вершинами v_6 и v_5 восстанавливается ребро e_5. Ребра e_6 и e_4 наследуют зеленый цвет ребра e_{16}, а ребра e_{14} и e_5 раскрашиваются в два оставшихся цвета.

Второй шаг восстановления относится к редуцированию по уникальному ребру. Сначала ищется красно-синий цикл, содержащий ребро e_{14}. Так как он не содержит ребро e_{15}, цикл из двух ребер e_{14}-e_5 перекрашивается. После этого ищется сине-зеленый цикл, содержащий ребро e_{14}. Это цикл e_{14}-e_6-e_{15}-e_4, и он содержит ребро e_{15}. Происходит восстановление ребра по найденному двухцветному циклу.

Третий шаг восстановления также относится к редуцированию по уникальному ребру. Так как существует сине-красный цикл e_{12}-e_2-e_{13}-e_5-e_6-e_7, сразу можно выполнить восстановление ребра по этому циклу (в данном случае можно выполнить восстановление и по сине-зеленому циклу, что приведет к другой раскраске).

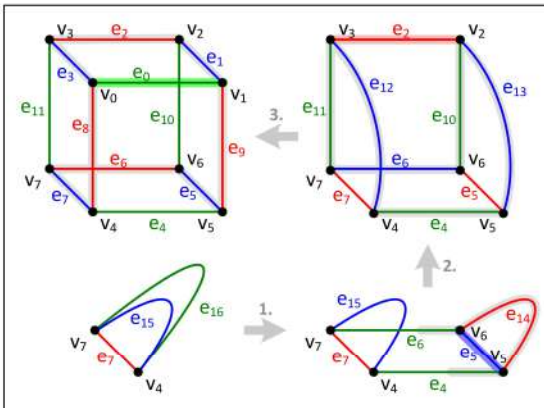

Рис. 8. Восстановление и раскраска графа

Fig. 8. Restoring and painting a graph

Реализацию данного алгоритма можно найти в репозитории https://github.com/r-aax/caesar (метод graph::Graph:: edges_coloring_for_cubic_graph_with_bicolor_cycles_algorithm).

Очевидно, что описанный алгоритм имеет квадратичную сложность по порядку графа, так как количество шагов восстановления пропорционально порядку исходного графа, а на каждом шаге восстановления уникального ребра необходимо выполнять поиск и перекраску двухцветных циклов, что в худшем случае пропорционально порядку текущего графа.

Анализ полученных результатов

Если рассматривать кубический граф порядка n, то он содержит $3n/2$ ребер, причем n четно. Если данный граф допускает правильную реберную раскраску в три цвета, то в каждый из цветов раскрашено ровно $n/2$ ребер. Таким образом, множество ребер распадается на три одинаковых по размеру подмножества ребер без конфликтов. Вызывает интерес вопрос о процентном соотношении ребер, раскрашенных в разные цвета, для жадной раскраски в пять цветов при порядке графа, стремящемся в бесконечность.

Для экспериментального получения такого распределения будем строить искусственные кубические графы следующим образом. В качестве нулевого графа возьмем кубический граф K_4, далее последовательно в текущем кубическом графе будем случайным образом выбирать вершину и заменять ее на треугольную конструкцию, как показано на рисунке 9. Вполне очевидно, что получающиеся таким образом графы будут кубическими и допускают правильную реберную раскраску в три цвета.

К полученным сгенерированным описанным способом кубическим графам порядка более 10^5 был применен жадный линейный алгоритм реберной раскраски в пять цветов. Получившееся распределение цветов показано на рисунке 10.

На рисунке видно, что распределение цветов жадной раскраски устроено примерно следующим образом. Большинство ребер кубического графа практически равномерно раскрасились в три цвета, однако для окрашивания около 10 % ребер потребовалось использование четвертого цвета, а для окрашивания около 1,5 % ребер пришлось задействовать пятый цвет.

Для оценки целесообразности применения реберной раскраски для устранения конфликтов по данным при расчете потоков в задаче моделирования процесса обледенения поверхности были выполнены запуски модельной задачи на одном вычислительном узле на базе микропроцессора Intel Xeon Phi Knights Landing 7 290. Данный микропроцессор имеет 72 ядра, на каждом из которых может быть запущено до четырех потоков. Таким образом, на этом микропроцессоре возможно распараллеливание запуска на 288 потоков [16].

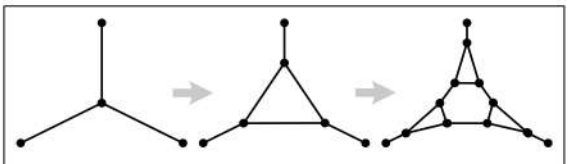

Рис. 9. Генерация кубического графа

Fig. 9. Cubic graph generation

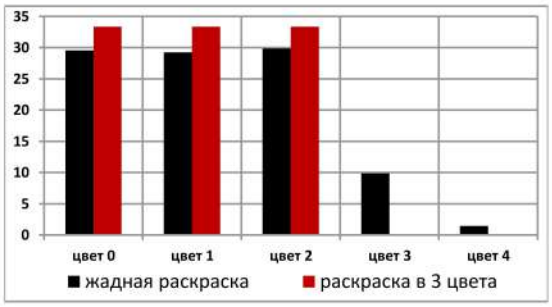

Рис. 10. Распределение процентной доли цветов для разных раскрасок

Fig. 10. Colors percentage distribution for different colorings

Для сравнения эффективности методов устранения конфликтов при параллельном расчете потоков через границы ячеек замерялась эффективность распараллеливания данного кода. Она считалась следующим образом. В качестве ускорения, достигаемого при распараллеливании на i потоков, бралась величина $s(i) = t(1)/t(i)$, где $t(1)$ – эталонное время вычислений при использовании одного потока, $t(i)$ – время вычислений при использовании i потоков. Под эффективностью распараллеливания понималась величина, вычисляемая как $e(i) = s(i)/i$. Смысл данной величины заключается в следующем. Можно предположить, что при идеальном распараллеливании вычислений при увеличении количества потоков ровно в k раз время выполнения уменьшается ровно в k раз. Таким образом, в случае идеального распараллеливания $s(i) = i$, а $e(i) = 1$. Эффективность распараллеливания вычислений является удобным показателем качества исполняемого

параллельного кода и сравнения различных вычислительных систем между собой.

На рисунке 11 можно заметить, что при использовании распараллеливания на относительно небольшое количество потоков нет существенной разницы между описанными методами устранения конфликтов. Однако при возрастании количества используемых потоков метод реберной раскраски гораздо эффективнее использования директивы omp atomic с точки зрения эффективности распараллеливания.

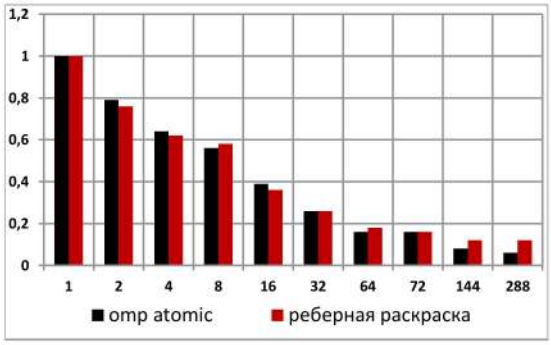

Рис. 11. Эффективность распараллеливания пересчета потоков для разных способов устранения конфликтов по данным

Fig. 11. Flows calculation parallelization efficiency for different data conflict elimination methods

Заключение

В процессе работы над оптимизацией ПО компьютерного моделирования обледенения поверхности выявлено узкое место для распараллеливания вычислений на общей памяти, связанное с возникновением конфликтов по данным. Эти конфликты приводят к необходимости использования режима атомарного исполнения отдельных команд внутри распараллеленного цикла, а также создают существенные препятствия для векторизации кода. Для устранения конфликтов рассмотрен и предложен альтернативный подход, основанный на построении реберной раскраски дуального графа расчетной сетки.

Для построения реберных раскрасок использованы два алгоритма: первый – тривиальный жадный линейный алгоритм раскраски в пять цветов, второй – квадратичный алгоритм построения раскраски Тейта, основанный на последовательном редуцировании и восстановлении графа. В ходе экспериментов выявлено, что при использовании большого количества потоков при распараллеливании использование реберной раскраски является более эффективным методом устранения конфликтов. Также следует отметить, что конкретный алгоритм раскраски графа не оказывает влияния на эффективность расчетов, то есть на практике может быть применим простой алгоритм раскраски в пять цветов.

Список литературы

1. Guzmán F. Finite volume methods. In: Numerical methods for initial value problems in physics, Springer Publ., 2023, pp. 205–258. doi: 10.1007/978-3-031-33556-3_5.

2. Tey W.Y., Asako Y., Wong K.Y. Comparison study between Galerkin finite element method and finite volume method for diffusion problem. J. of Advanced Research in Numerical Heat Transfer, 2024, vol. 15, no. 1, pp. 24–42. doi: 10.37934/arnht.15.1.2442.

3. Sun J.-K., Lin Ch.-D., Su X.-L., Tan Z.-Ch., Chen Y.-L., Ming P.-J. Solution of the discrete Boltzmann equation: Based on the finite volume method. Acta Phys. Sin., 2024, vol. 73, no. 11, art. 110504. doi: 10.7498/aps.73.20231984.

4. Río-Martín L., Busto S., Dumbster M. Massively parallel hybrid finite volume/finite element scheme for computational fluid dynamics. Math., 2021, vol. 9, no. 18, art. 2316. doi: 10.3390/math9182316.

5. Afzal A., Saleel C.A., Prashantha K., Bhattacharyya S., Sadhikh M. Parallel finite volume method-based fluid flow computations using OpenMP and CUDA applying different schemes. J. of Thermal Analysis and Calorimetry, 2021, vol. 145, pp. 1891–1909. doi: 10.1007/s10973-021-10637-1.

6. Tsoutsanis P., Antoniadis A.F., Jenkins K.W. Improvement of the computational performance of a parallel unstructured WENO finite volume CFD code for Implicit Large Eddy Simulation. Computers & Fluids, 2018, vol. 173, pp. 157–170. doi: 10.1016/j.compfluid.2018.03.012.

7. Брыков Н., Волков К., Емельянов В. Использование векторизованных структур данных при реализации вычислительных алгоритмов решения задач механики сплошной среды // Науч.-технич. вестн. информационных технологий, механики и оптики. 2022. Т. 22. № 1. С. 193–205. doi: 10.17586/2226-1494-2022-22-1-193-205.

8. Рыбаков А.А., Мещеряков А.О. Векторизация трехмерного метода погруженных границ для повышения эффективности расчетов на микропроцессорах Intel // Программные продукты и системы. 2023. Т. 36. № 1. С. 130–143. doi: 10.15827/0236-235X.141.130-143.

9. Mandel S. Predicting the formation of ice on aircraft. Scilight, 2019, vol. 2019, no. 52, art. 521108. doi: 10.1063/10.0000503.

10. Martini F., Contreras L.T., Ilinca A. Review of wind turbine icing modelling approaches. Energies, 2021, vol. 14, no. 16, art. 5207. doi: 10.3390/en14165207.

11. Ignatowicz K., Morency F., Beaugendre H. Numerical simulation of ice accretion using Messinger-based approach: Effects of surface roughness. Proc. CASI Aero Conf., 2019, art. hal-02409011.

12. Ignatowicz K., Morency F., Beaugendre H. Sensitivity study of ice accretion simulation to roughness thermal correction model. Aerospace, 2021, vol. 8, no. 3, art. 84. doi: 10.3390/aerospace8030084.

13. Гильфанов Л.Л., Мигранов С.В., Бикбов А.А. Распараллеливание решения задач с использованием раскраски графов // Молодой ученый. 2021. № 5. С. 4–6.

14. Курапов С.В., Давидовский М.В., Толок А.В. Визуальный алгоритм раскраски плоских графов // Научная визуализация. 2018. Т. 10. № 3. С. 1–33. doi: 10.26583/sv.10.3.01.

15. Курапов С.В., Давидовский М.В., Толок А.В. Алгебраические методы раскраски кубических графов // Научная визуализация. 2020. Т. 2020. № 2. С. 21–36. doi: 10.26583/sv.12.2.03.

16. Gulicheva A. Parallelization of finite-volume numerical methods of computational fluid dynamics by means of shared memory computing systems. In: CCIS. Proc. HPCST, 2024, vol. 1986, pp. 105–116. doi: 10.1007/978-3-031-51057-1_8.

Software & Systems doi: 10.15827/0236-235X.142.374-383 2024, 37(3), pp. 374–383

Cubic graph edge coloring in the problem of calculation parallelization on an unstructured surface computational grid

Anastasiya A. Gulicheva [1], Alexey A. Rybakov [2, 3]✉

[1] MIREA – Russian Technological University, Moscow, 119454, Russian Federation
[2] Udmurt State University, Izhevsk, 426034, Russian Federation, Russian Federation
[3] JSCC RAS – branch of SRISA RAS, Moscow, 119334, Russian Federation

For citation
Gulicheva, A.A., Rybakov, A.A. (2024) 'Cubic graph edge coloring in the problem of calculation parallelization on an unstructured surface computational grid', *Software & Systems*, 37(3), pp. 374–383 (in Russ.). doi: 10.15827/0236-235X.142.374-383

Article info
Received: 06.05.2024 After revision: 10.06.2024 Accepted: 14.06.2024

Abstract. The paper considers the issue of improving the performance of finite-volume numerical methods on shared-memory computing systems. In these methods, data conflicts are possible at the stage of calculation of conservative value flows across computational cell boundaries. This leads to performance degradation, especially with a large number of simultaneously operating flows. To eliminate data conflicts, the paper proposes a solution based on partitioning the set of processed cell boundaries of the computational grid into subsets without conflicts and processing these sets separately. The authors consider the solution on the example of calculations on surface unstructured computational grids. For them the problem is reduced to the problem of constructing a cubic graph edge coloring. To construct a cubic graph edge coloring, the authors apply two algorithms: the trivial linear algorithm of coloring in five colors and the algorithm of Tait coloring in three colors. The authors of the paper compare two coloring algorithms, as well as measure the influence of the proposed approach on the efficiency of calculation parallelization on a surface computational grid. The use of the data dependency elimination approach using cubic graph edge coloring was tested on the numerical problem of calculating body surface icing. Specialists performed launches on an Intel Xeon Phi microprocessor with a large number of parallel flows. The results showed that when the number of flows increases up to 144 and more, the efficiency of parallelization using edge coloring is twice as high as that of the usual approach of dependency elimination using OpenMP directives.

Keywords: numerical methods, parallelization, vectorization, computational grid, dual graph, edges coloring, Tait coloring
Acknowledgements. The work was performed within the framework of the SIC «Kurchatov Institute» state assignment on the topic FNEF 2024 0016

References

1. Guzmán, F. (2023) 'Finite volume methods', in *Numerical Methods for Initial Value Problems in Physics*, Springer Publ., pp. 205–258. doi: 10.1007/978-3-031-33556-3_5.

2. Tey, W.Y., Asako, Y., Wong, K.Y. (2024) 'Comparison study between Galerkin finite element method and finite volume method for diffusion problem', *J. of Advanced Research in Numerical Heat Transfer*, 15(1), pp. 24–42. doi: 10.37934/arnht.15.1.2442.

3. Sun, J.-K., Lin, Ch.-D., Su, X.-L., Tan, Z.-Ch., Chen, Y.-L., Ming, P.-J. (2024) 'Solution of the discrete Boltzmann equation: Based on the finite volume method', *Acta Phys. Sin.*, 73(11), art. 110504. doi: 10.7498/aps.73.20231984.

4. Río-Martín, L., Busto, S., Dumbster, M. (2021) 'Massively parallel hybrid finite volume/finite element scheme for computational fluid dynamics', *Math*., 9(18), art. 2316. doi: 10.3390/math9182316.

5. Afzal, A., Saleel, C.A., Prashantha, K., Bhattacharyya, S., Sadhikh, M. (2021) 'Parallel finite volume method-based fluid flow computations using OpenMP and CUDA applying different schemes', *J. of Thermal Analysis and Calorimetry*, 145, pp. 1891–1909. doi: 10.1007/s10973-021-10637-1.

6. Tsoutsanis, P., Antoniadis, A.F., Jenkins, K.W. (2018) 'Improvement of the computational performance of a parallel unstructured WENO finite volume CFD code for Implicit Large Eddy Simulation', *Computers & Fluids*, 173, pp. 157–170. doi: 10.1016/j.compfluid.2018.03.012.

7. Brykov, N.A., Volkov, K.N., Emelyanov, V.N. (2022) 'Vectorized numerical algorithms for the solution of continuum mechanics problems', *Sci. Tech. J. Inf. Technol. Mech. Opt*., 22(1), pp. 193–205 (in Russ.). doi: 10.17586/2226-1494-2022-22-1-193-205.

8. Rybakov, A.A., Meshcheryakov, A.O. (2023) 'Vectorization of the three-dimensional immersed boundary method for improving the efficiency of calculations on Intel microprocessors', *Software & Systems*, 36(1), pp. 130–143 (in Russ.). doi: 10.15827/0236-235X.141.130-143.

9. Mandel, S. (2019) 'Predicting the formation of ice on aircraft', *Scilight*, 2019(52), art. 521108. doi: 10.1063/10.0000503.

10. Martini, F., Contreras, L.T., Ilinca, A. (2021) 'Review of wind turbine icing modelling approaches', *Energies*, 14(16), art. 5207. doi: 10.3390/en14165207.

11. Ignatowicz, K., Morency, F., Beaugendre, H. (2019) 'Numerical simulation of ice accretion using Messinger-based approach: Effects of surface roughness', *Proc. CASI Aero Conf*., art. hal-02409011.

12. Ignatowicz, K., Morency, F., Beaugendre, H. (2021) 'Sensitivity study of ice accretion simulation to roughness thermal correction model', *Aerospace*, 8(3), art. 84. doi: 10.3390/aerospace8030084.

13. Gilfanov, L.L., Migranov, S.V., Bikbov, A.A. (2021) 'Parallelization of problem solving using graph coloring', *Young Sci*., (5), pp. 4–6 (in Russ.).

14. Kurapov, S.V., Davidovsky, M.V., Tolok, A.V. (2018) 'Visual algorithm for coloring planar graphs', *Sci. Visualization*, 10(3), pp. 1–33 (in Russ.). doi: 10.26583/sv.10.3.01.

15. Kurapov, S.V., Davidovsky, M.V., Tolok, A.V. (2020) 'Algebraic methods for coloring cubic graphs', *Sci. Visualization*, 2020(2), pp. 21–36 (in Russ.). doi: 10.26583/sv.12.2.03.

16. Gulicheva, A. (2024) 'Parallelization of finite-volume numerical methods of computational fluid dynamics by means of shared memory computing systems', in *CCIS. Proc. HPCST*, 1986, pp. 105–116. doi: 10.1007/978-3-031-51057-1_8.

Авторы

Гуличева Анастасия Алексеевна [1],
ассистент, gulicheva@mirea.ru
Рыбаков Алексей Анатольевич [2, 3],
к.ф.-м.н., начальник отдела,
ведущий научный сотрудник,
rybakov@jscc.ru

[1] МИРЭА – Российский технологический университет, г. Москва, 119454, Россия
[2] Национальный исследовательский центр «Курчатовский институт», г. Москва, 123182, Россия
[3] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН г. Москва, 119334, Россия

Authors

Anastasiya A. Gulicheva [1],
Assistant, gulicheva@mirea.ru
Alexey A. Rybakov [2, 3],
Cand. of Sci. (Physics and Mathematics),
Head of Department, Leading Researcher,
rybakov@jscc.ru

[1] MIREA – Russian Technological University, Moscow, 119454, Russian Federation
[2] National Research Centre "Kurchatov Institute", Moscow, 123182, Russian Federation
[3] JSCC RAS – branch of SRISA RAS, Moscow, 119334, Russian Federation