УДК 519.658; 519.677; 519.176

ОПТИМИЗАЦИЯ АЛГОРИТМА РАЗБИЕНИЯ ГИПЕРГРАФА С ПРОИЗВОЛЬНЫМИ ВЕСАМИ ВЕРШИН

А. С. Русаков[1], **М. В. Шеблаев**[2]

Одним из способов декомпозиции задачи большой размерности на подзадачи является представление ее в виде графа или гиперграфа и последующее разбиение на приблизительно равные части, причем число связей между подграфами должно быть минимальным. Если веса ребер графа моделируют объем межпроцессорных связей, а вес узлов гиперграфа — вычислительную сложность, то подзадачи можно эффективно решать на параллельных вычислительных системах. Известные многоуровневые эвристические методы на базе алгоритма Фидуччиа–Мэтьюза позволяют решать такие задачи за приемлемое время. В настоящей статье предложены модификации ключевой структуры данных алгоритма, позволяющие улучшить свойства метода сбалансированного разбиения гиперграфа на подграфы с целью достижения меньшего размера сечения и уменьшения издержек параллельных методов решения исходной задачи. Приведены результаты сравнения нового алгоритма для одноуровневого и иерархического методов разбиения.

Ключевые слова: разбиение гиперграфа, FM-алгоритм, кластеризация, распределенные вычислительные системы, параллельное программирование.

1. Введение. При решении неструктурированных задач на распределенных вычислительных системах зачастую возникает необходимость разбиения исходной задачи большой размерности на подзадачи для того, чтобы решать каждую из них на выделенных узлах. Например, задача оптимального распределения по процессорам нерегулярных сеток сводится к разбиению на компактные подобласти графа, веса вершин которого отражают объемы вычислений в различных узлах сетки, а веса ребер — объемы обменов данными, передаваемых в процессе счета [1]. При этом требуется сбалансировать использование распределенных вычислительных ресурсов и минимизировать межпроцессорные коммуникации. Во многих случаях сложность балансировки ограничивает достижимое ускорение на параллельных системах [2, 3]. Если исходная задача может быть описана в виде графа или гиперграфа, то задача балансировки может быть решена с помощью алгоритмов сбалансированного разбиения гиперграфа [4–6]. Эта задача возникает и в других приложениях: например, при решении систем линейных алгебраических уравнений, в задачах дискретной математики, при проектировании электронных схем [7] и др.

Пусть $H(V, E)$ — гиперграф с множеством вершин V и множеством гиперребер E. Пусть для каждого ребра $e \in E$ определена весовая функция $w_e : E \to Z$, $w_e > 0$, а для каждой вершины гиперграфа $v \in V$ определена весовая функция $w_v : V \to R$. Можно считать, что вес ребра моделирует объем межпроцессорных связей, а вес узлов гиперграфа — вычислительную сложность.

Задача минимизации разбиения гиперграфа, удовлетворяющего условию баланса, состоит в построении таких непересекающихся подмножеств $A \subset V$, $B \subset V$, $V = A \cup B$, что для ребер e, принадлежащих разрезу $\Psi = \big((u, v) \in E : u \in A, v \in B\big)$, минимизируется функция стоимости

$$\mathrm{CutCost}(\Psi) = \sum_{e \in \Psi} w_e(e) \tag{1}$$

при условии выполнения балансовых ограничений

$$1 - \varepsilon \leqslant \frac{\sum\limits_{v \in A} w_v(v)}{\sum\limits_{v \in B} w_v(v)} \leqslant 1 + \varepsilon, \tag{2}$$

[1] Институт проблем проектирования в микроэлектронике РАН, ул. Советская, д. 3, 124365, Зеленоград; ст. науч. сотр., e-mail: rusakov@inm.ras.ru

[2] Институт проблем проектирования в микроэлектронике РАН, ул. Советская, д. 3, 124365, Зеленоград; науч. сотр., e-mail: sheblaev@gmail.com

где ε — допустимое расхождение суммы весов двух подмножеств. Типичные значения ε находятся в пределах от 1% до 5%.

Сформулированная задача принадлежит классу NP [8]. В настоящее время известны пакеты программ, использующие идею многоуровневого разбиения графа на подграфы, такие как (h)METIS, Scotch, Zoltan, MLPART и др. В этих пакетах реализована возможность декомпозиции графа на N подграфов или же предлагается рекурсивно разбивать граф задачи на два. Как показано в работе [9], для качественного решения задачи сбалансированного разбиения гиперграфа большой размерности необходима кластеризация вершин, делающая алгоритм многоуровневым.

Общая структура современных программ предполагает выполнение следующих шагов:

1) кластеризация исходной задачи,

2) разбиение кластеризованного графа сравнительно небольшой размерности,

3) развертывание кластеризованного графа с одновременным уточнением.

Для решения задачи разбиения графа или гиперграфа, которая возникает на шагах 2 и 3, используются эвристические алгоритмы. Среди первых предложений решения такой задачи был алгоритм Кернигана–Лина [10]. Этот алгоритм итеративно улучшал начальное разбиение графа за конечное число проходов. Сложность этого алгоритма составляет $O\big(|V|^3\big)$. Базовый принцип попарного обмена между элементами подмножеств был в дальнейшем применен в эвристической процедуре разбиения двудольного графа [11].

В настоящее время часто используются другие подходы, среди которых можно упомянуть как точные комбинаторные методы [12] и спектральные методы [13], так и развитие идеи пошагового итеративного улучшения — алгоритм Фидуччиа–Мэтьюза (Fiduccia–Mattheyses, FM-алгоритм) [14]. Этот алгоритм имеет сложность $O\big(|V|\big)$ и находит решение, близкое к оптимальному. Кроме того, он позволяет решать также и другие задачи, близкие по постановке к (1), (2), возникающие в приложениях, и является наиболее универсальным.

Классический FM-алгоритм применяется для графов с близкими весами вершин $w(v)$. Если веса вершин сильно отличаются, то известные реализации существенно теряют в качестве разбиения. Такой анализ был проведен, например, в работе [15]. Кроме того, даже если входной граф многоуровневого алгоритма состоит из вершин с одинаковыми весами, то кластеризованный граф теряет это свойство. В [16] предлагалось допускать временные нарушения условий выполнения балансовых ограничений. В работе [15] предложено использовать предварительные итерации FM-алгоритма с увеличенным значением ε.

В настоящей статье предлагаются две модификации алгоритма Фидуччиа–Мэтьюза. В первой мы усовершенствовали базовую структуру данных FM-алгоритма, чтобы улучшить поведение алгоритма для гиперграфов с большими весами ячеек. Во второй модификации предложено использовать комбинацию эвристик при выборе вершин с одинаковой стоимостью при перемещении между разбиениями, что также позволяет усовершенствовать поведение алгоритма.

Мы реализовали предложенные эвристики в рамках нашего многоуровневого алгоритма сбалансированного разбиения гиперграфа [17]. Экспериментальный анализ проведен на наборе известных тестовых гиперграфов [18]. Особенностью этих гиперграфов является наличие вершин со значительно различающимися весами.

2. Итеративный алгоритм сбалансированного разбиения. Классический FM-алгоритм состоит из последовательных итераций. Входной информацией для одной итерации алгоритма является разбиение множества вершин заданного гиперграфа на две непустые части V_1 и V_2. Разбиение обязано удовлетворять условию баланса (2) и может быть произвольным. Все вершины считаются незафиксированными. Для каждой вершины v гиперграфа определим

— "удерживающую силу" $TE(v)$ как сумму весов гиперребер, полностью содержащихся в той же компоненте разбиения, что и v, и инцидентных v;

— "перетягивающую силу" $FS(v)$ как сумму весов ребер разреза, инцидентных вершине v и неинцидентных другим вершинам из той же компоненты разбиения;

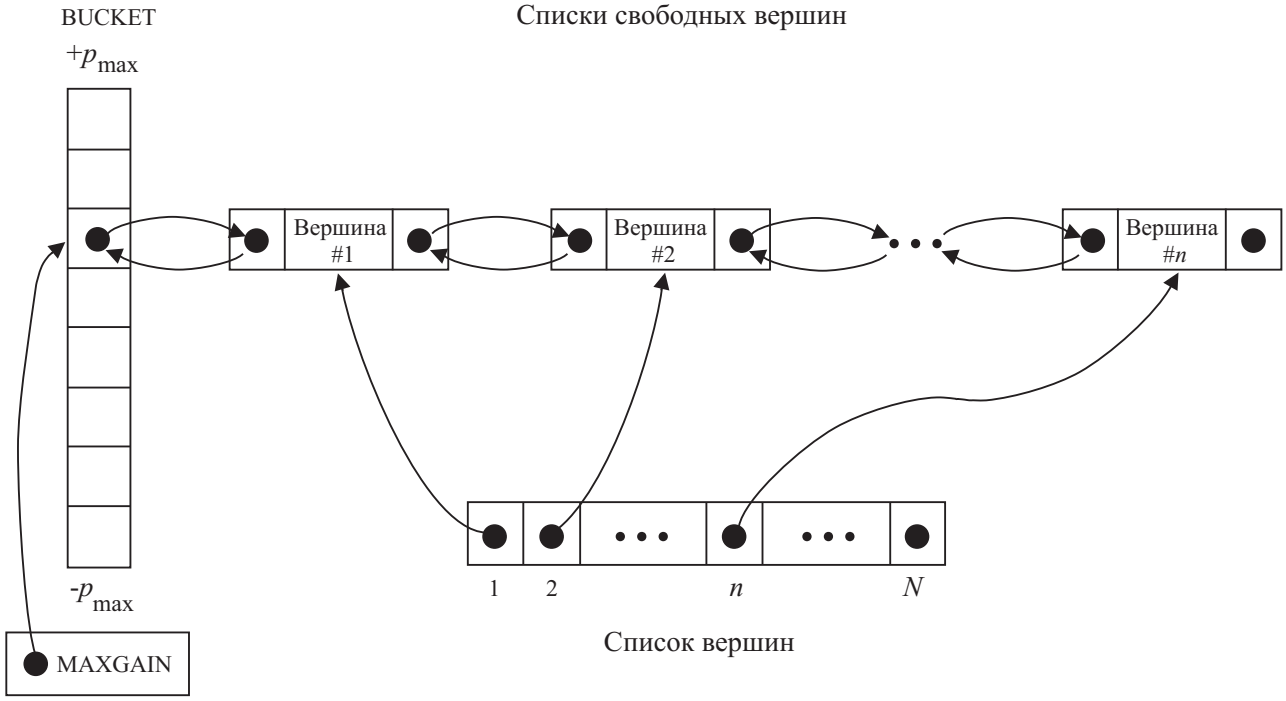
— стоимость $\Delta(v) = FS(v) - TE(v)$.

В одной итерации FM-алгоритма последовательно выбираются и перемещаются "базовые вершины". "Базовая вершина" — это незафиксированная вершина с наибольшей стоимостью, которую можно переместить в другую компоненту разбиения без нарушения условия баланса. Эта вершина перемещается и фиксируется. Стоимости всех инцидентных ей вершин пересчитываются, и выбирается новая базовая вершина.

Ключевым элементом FM-алгоритма является "корзина движений" — структура данных для хранения и быстрого пересчета стоимости перемещений вершин между A и B, позволяющая найти перемещение с наибольшей стоимостью со сложностью $O(1)$. Простейшая реализация корзины движений представляет

собой массив списков, в котором каждому возможному значению стоимости $\Delta(v)$ соответствует список вершин, позволяющий выполнять операции вставки/удаления со сложностью $O(1)$. Корзина движений снабжена индексным массивом, который содержит указатели на элементы списков, позволяющие получить позицию заданной вершины в массиве списков. Так как веса ребер w_e целые, то стоимость движения $\Delta(v)$ можно использовать для индексации массива. Отметим, что это ограничение не является принципиальным и существуют реализации корзины движений, поддерживающие $w_e \in R$. Таким образом, для того чтобы найти базовую вершину, достаточно начать линейный проход по спискам, начиная с наибольшего значения $\Delta(v)$. Вычисление измененной стоимости вершины и инцидентных ей выполняется за время, пропорциональное числу связей вершины, в большинстве случаев можно считать, что за время $O(1)$.

На рисунке приведена схема корзины движений. Элементы в массиве BUCKET индексированы значениями от $-p_{\max}$ до $p_{\max}$, где $p_{\max}$ — максимальная стоимость вершины в гиперграфе. Кроме того, структура данных содержит указатель MAXGAIN на список вершин с максимальной стоимостью.



Итерации повторяются заданное число раз или до тех пор, пока не наступает насыщение алгоритма. Псевдокод одной итерации FM-алгоритма представлен в виде алгоритма 1.

Алгоритм 1. Итерации FM-алгоритма.

function FM Pass (graph $G = (V, E)$, max balance $maxbal$)
  $P_{\text{saved}} \leftarrow V_1, V_2$ ▷ Начальное разбиение $V = V_1 \cup V_2$
  $\text{CutCost}_{\text{saved}} \leftarrow \text{CutCost}(P_{\text{saved}})$
  Вычислить $\Delta(v_i)$ для каждой $v_i \in V$
  repeat
    $v = \text{GetBestMove}$ ▷ Выбрать незафиксированную вершину $v \in V$ с максимальной $\Delta(v)$
    if $v \in V_1$ **then**
      $V_1 \leftarrow V_1 - v$; $V_2 \leftarrow V_2 + v$
    else
      $V_2 \leftarrow V_2 - v$; $V_1 \leftarrow V_1 + v$
    end if
    for v', связанных ребром с v **do**
      if v' не зафиксирована **then**
        Вычислить $\Delta(v')$
        $\text{UpdateGainCell}\Delta(v')$ ▷ Обновить стоимость вершины в корзине
      end if
    end for
    Зафиксировать v

```
            P ← V1, V2
            if CutCost_saved > CutCost(V1, V2) then
                CutCost_saved ← CutCost(V1, V2)
                P_saved ← V1, V2
            end if
        until есть доступные базовые вершины
        return P_saved
    end function
```

Численные эксперименты проведены в рамках предложенной в [17] программной реализации FM-алгоритма и многоуровневого метода разбиения. Согласно подходу многоуровневого разбиения, реализация [17] состоит из следующих этапов:

1) кластеризация гиперграфа (coarsening);

2) нахождение нескольких лучших первоначальных разбиений кластеризованного графа (initial partition);

3) развертывание кластеризованного графа и оптимизация разбиения на каждом уровне для выбранных решений (refinement), выбор лучшего решения.

Кластеризацией гиперграфа называется процесс построения гиперграфа, содержащего меньшее число узлов, чем исходный. К полученному графу опять применяется процедура кластеризации, и в итоге мы получаем иерархию гиперграфов, состоящую из нескольких уровней. В [17] рассмотрены известные алгоритмы кластеризации HEC (Hyper Edge Coarsening), MHEC (Modified Hyper Edge Coarsening) [9] и HEM (Hierarchical Expectation Maximization), FC (FirstChoice) [19]. При сравнении алгоритмов кластеризации мы обнаружили, что алгоритмы HEC и MHEC достаточно быстро достигают насыщения. Размерность кластеризованного гиперграфа оказывается слишком большой, и поиск минимального сечения такого графа оказывается неэффективным. Чтобы улучшить вычислительную эффективность алгоритма, к гиперграфу, полученному из HEC или MHEC, мы применяем метод кластеризации FC, который успешно сжимает гиперграф еще в несколько раз. Результат применения эвристики существенно зависит от входного разбиения. Входное разбиение для FM-алгоритма на этапе 2 может выбираться случайно, в нашей работе мы случайно назначали несколько вершин в подобласти и с помощью поиска в ширину от этих вершин назначали оставшиеся вершины графа, чтобы получить входное разбиение. На этапе 2 мы находили решения 35 разных вариантов. Из этих решений 10 лучших оптимизировались при развертывании графа. Такой процесс повторялся для каждого из применямых методов кластеризации, а лучшее разбиение выбиралось в качестве итогового решения. Мы используем также идею из [15]: при начальном разбиении на первых нескольких итерациях FM-алгоритма принимается завышенное значение $\varepsilon = 10\%$. На каждой третьей итерации мы применяем CLIP-версию FM-алгоритма [19].

3. Модификация корзины движений. Корзина движений позволяет найти базовую ячейку за время $O(1)$. Если в исходном или в кластеризованном гиперграфе веса вершин $w(v)$ значительно варьируются, то при поиске оптимального движения возникают трудности. Если в графе много вершин с большим весом и большим количеством связей, то эти вершины оказываются сверху корзины, но их движение нарушает условие баланса и поиск сбалансированной оптимальной ячейки может потребовать существенно больше операций. В этом случае не существует однозначно лучшего подхода, который обеспечивал бы как минимизацию разреза, так и быстродействие алгоритма. Известные алгоритмы поиска движений в корзине часто оказываются неэффективными, так как движения для большого количества ячеек могут нарушать балансовые ограничения. Существуют разные стратегии обработки таких ячеек. Самая простая — удалять несбалансированные узлы гиперграфа из корзины, как только они встречаются. Такой узел двигаться на данной итерации FM-алгоритма не будет. Вторая стратегия — не удалять узлы графа, которые на данном шаге не сбалансированы, а оставлять их в корзине и передвигать в другую подобласть, как только условие баланса будет достигнуто и узел будет узлом максимальной стоимости. Псевдокод этих стратегий приведен на схемах алгоритмов 2 и 3.

Алгоритм 2. Классический быстрый поиск в корзине движений.

```
function GetBestMove
    cell ← maxGainCell ▷ поиск следующей сбалансированной вершины начинается с узла с
                       ▷ наибольшей ценой в корзине движений
    while cell != NULL do
        if IsBalancedMove(cell) then ▷ Проверка выполнения балансовых ограничений
            RemoveCellFromBucket (cell)
            return cell
```

Таблица 1

Размер сечения и время работы многоуровневого алгоритма разбиения гиперграфа с разными корзинами FM-алгоритма, $\varepsilon = 0.02$

	Медленный поиск			Классический поиск			Новый поиск		
Тест	Размер сечения	Время, с	Качество	Размер сечения	Время, с	Качество	Размер сечения	Время, с	Качество
ibm01	219	4	1	336	3	0.65	245	4	0.89
ibm02	269	5	0.99	266	5	1	269	5	0.99
ibm03	751	10	1	911	7	0.82	791	8	0.95
ibm04	535	9	0.96	513	7	1	515	8	1
ibm05	1733	11	1	1725	10	1	1732	12	1
ibm06	585	11	0.89	788	11	0.66	535	11	0.98
ibm07	802	22	1	820	15	0.98	833	17	0.96
ibm08	1190	32	1	1187	16	1	1190	17	1
ibm09	533	18	1	535	16	1	535	17	1
ibm10	1119	39	0.97	1216	17	0.89	1084	17	1
ibm11	874	34	0.97	844	27	1	868	26	0.97
ibm12	2195	64	0.95	2084	26	1	2185	33	0.95
ibm13	1073	62	0.95	1024	43	1	1111	43	0.92
ibm14	1938	100	1	1960	74	0.99	1955	77	0.99
ibm15	2567	200	1	2709	93	0.95	2653	92	0.97
ibm16	1788	151	1	1868	104	0.96	1805	92	0.99
ibm17	2296	150	1	2648	92	0.87	2362	158	0.97
ibm18	1943	49	1	2057	23	0.94	2232	40	0.87
Score		971	17.67		589	16.71		677	17.4

```
            end if
            RemoveCellFromBucket (cell) ▷ вершина удаляется из корзины, как только
                                        ▷ она рассмотрена
            cell ← cell− > next
        end while
        return NULL
    end function
```

Алгоритм 3. Медленный поиск в корзине движений.

```
    function GetBestMove
        cell ← maxGainCell
        while cell != NULL do
            if IsBalancedMove(cell) then ▷ Проверка выполнения балансовых ограничений
                cell ← cell− > next ▷ вершина удаляется из корзины только после ее движения
                RemoveCellFromBucket (cell)
                return cell
            end if
        end while
        return NULL
    end function
```

В табл. 1 и 2 мы приводим результаты запусков FM-алгоритма с разными модификациями стратегий на тестах [18], показывающие, что применение алгоритма 2 приводит к большему размеру сечений гипер-

графов по сравнению с алгоритмом 3. При этом даже увеличение числа итераций или числа начальных решений не позволяет приблизиться к решению, полученному при стратегии, в которой из корзины не выбрасываются несбалансированные ячейки. В то же время стратегия сохранения несбалансированных ячеек приводит к потере быстродействия алгоритма.

Таблица 2

Размер сечения и время работы неиерархического FM-алгоритма, $\varepsilon = 0.02$

	Медленный поиск			Классический поиск			Новый поиск		
Тест	Размер сечения	Время, с	Качество	Размер сечения	Время, с	Качество	Размер сечения	Время, с	Качество
ibm01	303	1	1	328	1	0.92	393	2	0.77
ibm02	446	4	0.84	8	2	0.49	373	3	1
ibm03	1734	4	0.85	2513	3	0.59	1558	3	0.95
ibm04	1042	7	0.94	1423	4	0.69	1093	5	0.9
ibm05	2010	7	0.93	3201	4	0.59	1878	5	1
ibm06	1076	9	0.96	2752	4	0.38	1173	7	0.88
ibm07	1505	16	0.89	1574	9	0.85	1332	12	1
ibm08	1937	35	1	5242	10	0.37	2491	17	0.78
ibm09	1246	20	1	3806	12	0.33	1457	15	0.86
ibm10	2638	49	0.85	4204	17	0.53	2230	24	1
ibm11	2612	32	0.82	5757	18	0.37	2136	22	1
ibm12	3722	73	1	8350	18	0.45	4741	30	0.79
ibm13	2188	39	0.76	3042	22	0.54	2036	28	0.81
ibm14	3855	69	0.82	7528	43	0.42	4561	54	0.7
ibm15	4948	168	0.87	9542	51	0.45	5246	75	0.82
ibm16	3683	179	1	10828	57	0.34	4356	82	0.85
ibm17	3285	117	1	13236	60	0.25	5434	79	0.6
ibm18	3345	191	1	9522	63	0.35	3758	102	0.89
Score		1020	16.52		398	8.89		565	15.58

Чтобы приблизиться к качеству медленной корзины движений, мы предложили модификацию с рестартами. В структуру данных добавлен указатель, который указывает либо на NULL, либо на следующую сбалансированную вершину. Изначально он инициализируется нулем. Как только мы находим узел графа с максимальной стоимостью и выбираем его для движения, указатель показывает на следующий элемент в корзине движений. Если указатель равен нулю, то поиск движения начнется с вершины максимальной стоимости, иначе поиск сбалансированной ячейки начнется с ячейки, маркированной указателем. Если после нескольких ходов алгоритма какая-либо ячейка из корзины приобретает стоимость большую, чем ячейка, на которую указывает наш указатель на сбалансированный элемент, мы его обнуляем, т.е. “делаем рестарт”. Таким образом, мы уменьшаем среднюю стоимость поиска лучшей ячейки для движения в другую подобласть, так как не нужно пробегать через длинный список ячеек с большим весом (или площадью), оказавшихся сверху корзины. В худшем случае такой алгоритм поиска все равно потребует $O(|V|)$ операций, а реализация FM-алгоритма требует $O(|V|^2)$ операций, но можно ожидать, что следующая ячейка для движения будет найдена за время порядка $O(1)$. Псевдокод поиска с рестартами представлен в виде алгоритмов 4 и 5.

Алгоритм 4. Новый поиск в корзине движений.

```
function GetBestMove
    cell ← maxGainCell
    if firstBalanced != NULL then
```

```
        cell ← firstBalanced ▷ начинаем поиск с предположительно сбалансированной вершины
    end if
    while cell != NULL do
        if IsBalancedMove(cell) then
            RemoveCellFromBucket (cell)
            firstBalanced ← cell− > next ▷ сохраняем указатель на вершину, следующую
                                         ▷ за сбалансированной
            return cell
        end if
    end while
    firstBalanced ← NULL
    return NULL
end function
```

Алгоритм 5. Рестарт корзины при модификации соседних узлов.

```
function UpdateGainCell newGain
    ...
    if newGain > firstBalanced− > gain then ▷ следующий поиск лучшей вершины начнется с
                                            ▷ верха корзины
        firstBalanced = NULL
    end if
    ...
end function
```

Экспериментальное сравнение новой структуры корзины движений для неиерархического (без кластеризации) и многоуровневого алгоритма приведено в табл. 1 и 2. Для анализа поведения нескольких алгоритмов на множестве тестов мы ввели коэффициент качества. Для каждого из N тестов мы находим лучшее значение сечения $BestCut$. Качество для каждого из тестов принимается в виде $q = BestCut/Cut$, где Cut — значение сечения на данном тесте для данного алгоритма. Суммарное качество алгоритма на всем множестве тестов вычисляется как $Q = \sum q$. Чем больше значение Q, тем меньше размер сечения и результат алгоритма для набора тестов лучше.

Результаты показывают, что применение корзины с рестартом позволяет существенно ускорить FM-алгоритм с “медленной”, но точной корзиной, при этом качество ухудшается или, что тоже самое, — размер сечения увеличивается, но незначительно. В многоуровневой реализации результат заметен меньше, так как FM-алгоритм применяется к кластеризованному графу меньшей размерности. Важно отметить, что такая модификация корзины движений практически не ухудшает сложность поиска базовой вершины даже в том случае, когда веса вершины одинаковые.

4. Эвристики выбора из вершин с одинаковой стоимостью. В оригинальном FM-алгоритме для каждого движения между подобластями выбирается вершина с наибольшей стоимостью. В том случае, когда таких вершин для очередного хода несколько, необходимо выбрать одну из них. Для этого выбора не существует доказанной оптимальной стратегии, но экспериментально показано, что разные стратегии существенно влияют на результат работы алгоритма. В работе [20] рассматривались следующие стратегии локального выбора:

— LIFO (Last In, First Out), при которой элемент, помещаемый в соответствующую корзину движений последним, выбирается первым;

— FIFO (First In, First Out), при которой элемент, пришедший первым, первым же и выбирается;

— Random, при которой реализуется случайный выбор элемента среди прочих кандидатов.

В работах [20, 21] экспериментально показано, что стратегия LIFO обеспечивает лучшее качество итогового разбиения среди перечисленных стратегий. Объяснение, предложенное авторами, состояло в том, что в этой стратегии вершины с сильной связностью (ядра кластеров) будут перемещаться последовательно и с бóльшим приоритетом.

В работе [22] авторы сделали наблюдение о том, что большее влияние может иметь перемещение вершин, связанных с недавно перемещенными вершинами. В таком случае предлагается использовать стратегию, разделяющую изменения стоимостей при перемещении на три типа:

— тип A, при котором стоимость увеличивается,

— тип B, при котором стоимость уменьшается,

— тип C, при котором стоимость остается неизменной.

Перемещения типа A, следуя анализу [20] для FM-алгоритма, должны поощряться и обрабатываться в первую очередь; перемещения типа B, напротив, должны влиять на последующие перемещения как можно меньше. В соответствии с этим наблюдением вершины при перемещении типа A должны помещаться в "активную зону" корзины движений, из которой они будут извлечены вскоре, т.е. в начало списка, а вершины типа B — в конец списка, чтобы не "отравлять" последующие перемещения. Вершины типа C остаются на своем месте. Такая стратегия получила название LIFO* [22].

Таблица 3

Сравнение стратегий локального выбора по размеру сечения для многоуровневого алгоритма разбиения гиперграфа

Тест	LIFO, размер сечения	LIFO, качество	LIFO*, размер сечения	LIFO*, качество	LIFO + LIFO*, размер сечения	LIFO + LIFO*, качество
ibm01	262	0.94	252	0.97	245	1.0
ibm02	269	1.00	346	0.78	269	1.0
ibm03	826	1.00	850	0.97	822	1.0
ibm04	537	1.00	539	1.00	537	1.0
ibm05	1731	1.00	1726	1.00	1731	1.0
ibm06	596	1.00	718	0.83	596	1.0
ibm07	820	1.00	820	1.00	820	1.0
ibm08	1190	1.00	1190	1.00	1190	1.0
ibm09	535	1.00	535	1.00	535	1.0
ibm10	1088	0.93	1011	1.00	1088	1.0
ibm11	877	0.94	820	1.00	877	1.0
ibm12	2185	1.00	2193	1.00	2185	1.0
ibm13	1170	1.00	1175	0.99	1166	1.0
ibm14	1955	1.00	1969	0.99	1955	1.0
ibm15	2488	1.00	2639	0.94	2488	1.0
ibm16	1805	0.99	1824	0.98	1789	1.0
ibm17	2378	1.00	2403	0.99	2378	1.0
ibm18	1959	0.99	1931	1.00	1959	1.0
		17.77		17.44		18.0

Мы реализовали и сравнили разные стратегии включая LIFO* на примерах гиперграфов [18]. Мы получили в среднем результаты хуже по сравнению со стратегией LIFO. В то же время на некоторых гиперграфах новая эвристика работает лучше. Для того чтобы улучшить результаты нашего алгоритма, мы реализовали комбинацию эвристик LIFO* и LIFO. На каждой второй итерации FM-алгоритма мы используем LIFO*. Мы предполагаем, что комбинация эвристик эффективнее позволяет выйти из локального минимума. Результаты, представленные в табл. 3, показывают, что наше предложение позволило добиться заметного роста качества разбиения без увеличения времени работы.

5. Заключение. В настоящей статье предложены модификации эвристик FM-алгоритма для задачи разбиения гиперграфа с произвольными весами вершин. Экспериментально показано, что известные алгоритмы поиска "наилучшей" вершины в гиперграфе с произвольными весами вершин дают решение, далекое от оптимального. Новый алгоритм поиска в корзине движений позволяет получать сбалансированное разбиение гиперграфа с меньшим размером сечения без заметной потери быстродействия по сравнению с классическим FM-алгоритмом. Предложена комбинация стратегий разрешения неоднозначностей выбора в корзине движений LIFO* и LIFO, которая также позволила заметно улучшить качество алгоритма.

Статья рекомендована к публикации Программным комитетом научно-технического семинара "Тех-

нологии параллельной обработки больших графов" (GraphHPC-2014, http://www.dislab.org/GraphHPC-2014).

СПИСОК ЛИТЕРАТУРЫ

1. *Волков К.Н.* Балансировка нагрузки процессоров при решении краевых задач механики жидкости и газа сеточными методами // Вычислительные методы и программирование. 2012. **13**. 107–129.
2. *Hendrickson B., Kolda T.G.* Graph partitioning models for parallel computing // Parallel Computing. 2000. **26**, N 12. 1519–1534.
3. *Головченко Е.Н.* Комплекс программ параллельной декомпозиции сеток // Вычислительные методы и программирование. 2010. **11**. 360–365.
4. *Копысов С.П., Новиков А.К.* Параллельные алгоритмы адаптивного перестроения и разделения неструктурированных сеток // Математическое моделирование. 2002. **14**, № 9. 91–96.
5. *Якобовский М.В.* Инкрементный алгоритм декомпозиции графов // Вестн. Нижегородского ун-та им. Н. И. Лобачевского. Серия "Математическое моделирование и оптимальное управление". 2005. **28**, № 1. 243–250.
6. *Karypis G., Kumar V.* Multilevel k-way partitioning scheme for irregular graphs // Journal of Parallel and Distributed Computing. 1998. **48**, N 1. 96–129.
7. *Bichot C.E., Siarry P. (Eds.)* Graph partitioning. London–Hoboken: John Wiley & Sons, 2013.
8. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
9. *Karypis G., Kumar V.* A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1998. **20**, N 1. 359–392.
10. *Kernighan B.W., Lin S.* An efficient heuristic procedure for partitioning graphs // Bell System Technical Journal. 1970. **49**, N 1. 291–308.
11. *Русаков С.Г., Святский А.Б.* Метод автоматического разбиения на подсхемы для машинного анализа больших интегральных схем // Управляющие системы и машины. 1975. **3**. 116–119.
12. *Delling D., Goldberg A., Razenshteyn I., Werneck R.* Exact combinatorial branch-and-bound for graph bisection // Proc. Meeting on Algorithm Engineering and Experiments. Philadelphia: SIAM Press, 2012. 30–34.
13. *Pothen A., Simon D.H., Liou K.* Partitioning sparse matrices with eigenvectors of graphs // SIAM J. of Matrix Analysis and Applications. 1990. **11**, N 3. 430–452.
14. *Fiduccia C.M., Mattheyses R.M.* A linear-time heuristic for improving network partitions // Proc. of 19th IEEE Design Automation Conference. Piscataway: IEEE Press, 1982. 175–181.
15. *Caldwell A.E., Kahng A.B., Markov I.L.* Iterative partitioning with varying node weights // VLSI Design. 2000. **11**, N 3. 249–258.
16. *Bui-Xuan B.M., Telle J.A., Vatshelle M.* Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems // Theoretical Computer Science. 2013. **511**. 66–76.
17. *Андреев А.Е., Пависич И., Русаков А.С.* Алгоритм глобального размещения структурированных заказных схем // Проблемы разработки перспективных микро- и наноэлектронных систем. М.: ИППМ РАН, 2012. 231–236.
18. *Alpert C.J.* The ISPD98 circuit benchmark suite // Proc. of the 1998 ACM International Symposium on Physical Design. New York: ACM Press, 1998. 80–85.
19. *Papa D.A., Markov I.L.* Hypergraph partitioning and clustering // Handbook of Approximation Algorithms and Metaheuristics. Boca Raton: CRC Press, 2007. 61.1–61.19.
20. *Hagen L.W., Huang D.J.H., Kahng A.B.* On implementation choices for iterative improvement partitioning algorithms // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 1997. **16**, N 10. 1199–1205.
21. *Caldwell A.E., Kahng A.B., Markov I.L.* Improved algorithms for hypergraph bipartitioning // Proc. of the 2000 ACM Asia and South Pacific Design Automation Conference. New York: ACM Press, 2000. 661–666.
22. *Yoon Y., Kim Y.H.* New bucket managements in iterative improvement partitioning algorithms // Applied Mathematics & Information Sciences. 2013. **7**, N 2. 37–57.

Поступила в редакцию
02.06.2014

Optimization of a Partitioning Algorithm for a Hypergraph with Arbitrary Weights of Vertices

A. S. Rusakov [1] **and M. V. Sheblaev** [2]

[1] *Institute for Design Problems in Microelectronics, Russian Academy of Sciences; ulitsa Sovetskaya, 3, Zelenograd, 124365, Russia; Ph.D., Senior Scientist, e-mail: rusakov@inm.ras.ru*

[2] *Institute for Design Problems in Microelectronics, Russian Academy of Sciences; ulitsa Sovetskaya, 3, Zelenograd, 124365, Russia; Scientist, e-mail: sheblaev@gmail.com*

Received June 2, 2014

Abstract: One of the methods for the decomposition of a large problem to subproblems is its representation as a graph or hypergraph and partition this graph to approximately equal subgraphs with minimal cuts. The balanced hypergraph partitioning with the minimization of the cut size reduces communication cost between decomposed subproblems. The well-known approach to the hypergraph decomposition is the Fiduccia–Mattheyses (FM) algorithm and its hierarchical modifications. In this paper we discuss a key data structure modifications of the FM-algorithm to improve the performance and quality of the hierarchical partitioning algorithms and to reduce the computational overheads during solving large problems by parallel methods.

Keywords: hypergraph partitioning, Fiduccia–Mattheyses algorithm, clustering, distributed computing systems, parallel programming.

References

1. K. N. Volkov, "Load Balancing of Processors when Solving the Problems of Fluid and Gas Mechanics by Mesh Methods," Vychisl. Metody Programm. **13**, 107–129 (2012).

2. B. Hendrickson and T. G. Kolda, "Graph Partitioning Models for Parallel Computing," Parallel Comput. **26** (12), 1519–1534 (2000).

3. E. N. Golovchenko, "A Parallel Mesh Partitioning Tool," Vychisl. Metody Programm. **11**, 360–365 (2010).

4. S. P. Kopysov and A. K. Novikov, "Parallel Algorithms of Adaptive Refinement and Partitioning of Unstructured Grids," Mat. Model. **14** (9), 91–96 (2002).

5. M. V. Yakobovskii, "An Incremental Graph Decomposition Algorithm," Vestn. Lobachevsky State Univ. Nizhni Novgorod, Ser.: Mat. Model. Optim. Contr. **28** (1), 243–250 (2005).

6. G. Karypis and V. Kumar, "Multilevel k-Way Partitioning Scheme for Irregular Graphs," J. Parallel Distrib. Comput. **48** (1), 96–129 (1998).

7. C. E. Bichot and P. Siarry (Eds.), *Graph Partitioning* (Wiley, London, 2013).

8. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979; Mir, Moscow, 1982).

9. G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," SIAM J. Sci. Comput. **20** (1), 359–392 (1998).

10. B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," Bell Syst. Tech. J. **49** (1), 291–308 (1970).

11. S. G. Rusakov and A. B. Svyatskii, "A Method of Automated Partitioning into Subschemes for the Computerized Analysis of Integrated Circuits," Upravl. Syst. Mashiny **3**, 116–119 (1975).

12. D. Delling, A. Goldberg, I. Razenshteyn, and R. Werneck, "Exact Combinatorial Branch-and-Bound for Graph Bisection," in *Proc. Meeting on Algorithm Engineering and Experiments, Kyoto, Japan, January 16, 2012* (SIAM Press, Philadelphia, 2012), pp. 30–34.

13. A. Pothen, D. H. Simon, and K. Liou, "Partitioning Sparse Matrices with Eigenvectors of Graphs," SIAM J. Matrix Anal. Appl. **11** (3), 430–452 (1990).

14. C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in *Proc. 19th IEEE Design Automation Conference* (IEEE Press, Piscataway, 1982), pp. 175–181.

15. A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Iterative Partitioning with Varying Node Weights," VLSI Design **11** (3), 249–258 (2000).

16. B. M. Bui-Xuan, J. A. Telle, and M. Vatshelle, "Fast Dynamic Programming for Locally Checkable Vertex Subset and Vertex Partitioning Problems," Theor. Comput. Sci. **511**, 66–76 (2013).

17. A. E. Andreev, I. Pavisich, and A. S. Rusakov, "Global Placement Algorithm for Structured ASIC," in *Problems of Perspective Micro- and Nanoelectronic Systems Development* (Inst. for Design Problems in Microelectronics, Moscow, 2012), pp. 231–236.

18. C. J. Alpert, "The ISPD98 Circuit Benchmark Suite," in *Proc. 1998 ACM Int. Symp. on Physical Design* (ACM Press, New York, 1998), pp. 80–85.

19. D. A. Papa and I. L. Markov, "Hypergraph Partitioning and Clustering," in *Handbook of Approximation Algorithms and Metaheuristics* (CRC Press, Boca Raton, 2007), pp. 61.1–61.19.

20. L. W. Hagen, D. J. H. Huang, and A. B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms," IEEE Trans. Computer-Aided Design Integr. Circuits Syst. **16** (10), 1199–1205 (1997).

21. A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning," in *Proc. 2000 ACM Asia and South Pacific Design Automation Conf.* (ACM Press, New York, 2000), pp. 661–666.

22. Y. Yoon and Y. H. Kim, "New Bucket Managements in Iterative Improvement Partitioning Algorithms," Appl. Math. Inform. Sci. **7** (2), 37–57 (2013).