Содержание

Введение

Существует большое количество разнообразных технологий параллельного программирования. Разрабатываемое параллельное программное обеспечение предназначено только для конкретных архитектур вычислительных систем. В таком разнообразии бывает сложно подобрать наиболее эффективную вычислительную систему для решения конкретной задачи. Важным фактором в развитии параллельных вычислений является стоимость параллельных вычислительных систем. Дороговизна параллельных вычислительных систем, с одной стороны, тормозит развитие прикладных основ параллельных вычислений, а, с другой, дает мощный толчок теоретическим исследованиям в области параллельных вычислений. Но в любом случае разумно ориентироваться на существующие стандарты в параллельных вычислениях.

Обычно выделяют два вида параллелизма: *крупноблочный* [1] и *мелкозернистый* [2, 3]. Основное различие между этими двумя видами параллелизма заключается в отношении числа последовательно выполняемых команд к числу межпроцессорных обменов. Алгоритмы, основанные на крупноблочном параллелизме, выполняют (параллельно) небольшое число сложных последовательных вычислений с редкими обменами промежуточными результатами. Мелкозернистый параллелизм подразумевает большое количество параллельно выполняемых простых, часто одинаковых, вычислений, обмен результатами при этом происходит на каждом шаге [3]. Достоинства и недостатки есть у обоих видов параллелизма, подтверждают это и работы [3,4]. Параллельные вычислительные системы принято называть *многопроцессорными* или *суперЭВМ* [4].

В [4] предложена классификация наиболее распространенных многопроцессорных систем:

- *Векторно-конвейерные компьютеры*, характеризуются наличием функциональных конвейерных устройств и набором векторных команд.
- *Массивно-параллельные компьютеры* с распределенной памятью, представляют собой соединение микропроцессоров с помощью сетевого оборудования.

- *Параллельные компьютеры с общей памятью*, вся оперативная память этих машин разделяется несколькими одинаковыми процессорами, обращающимися к общей дисковой памяти. При этом проблем с обменом данными между процессорами и синхронизацией их работы практически не возникает.
- *Кластерные компьютеры*. Это класс, представляющий собой комбинацию предыдущих трех.

Параллельные машины, и, как следствие, параллельные вычисления, создаются для “быстрого” решения задач большой размерности. Использование индивидуальных особенностей отдельно взятой задачи существенно облегчить процесс ее распараллеливания, если говорить о сравнении с общими методами. Часто наилучший результат достигается при комбинированном использовании общепринятых методов и индивидуальных особенностей решаемой задачи.

Наиболее распространенным методом распараллеливания алгоритмов является метод, описанный в книге [5]. Широко используется этот метод и в распараллеливании алгоритмов решения теоретико-графовыех задач. О некоторых их них рассказано в следующем параграфе.

В настоящей работе проведено исследование, посвященное распараллеливанию алгоритмов на предфрактальных графах [6]. Предфрактальные графы – класс графов, обладающий рядом отличительных топологических и метрических свойств, с использованием которых и проводится распараллеливание алгоритмов.

Описанные в этой работе параллельные алгоритмы на предфрактальных графах относятся к категории крупноблочных и предполагают реализацию, по мнению авторов, на массивно-параллельных компьютерах.

Для оценки трудоемкости параллельных алгоритмов будем использовать хорошо известные критерии. Под *временем исполнения* алгоритма будем понимать число простых операций, произведенных за время работы алгоритма. *Ускорение* алгоритма представляет разницу между временем исполнения разработанного алгоритма и наилучшим временем исполнения известного последовательного алгоритма. *Вычислительная сложность* представляет общее число всех операций произведенных алгоритмом за время его работы.

1. Параллельные алгоритмы на графах

На настоящий момент в отечественной научной периодике нет работ связанных с распараллеливанием алгоритмов на графах. В оправдание можно сослаться на [7], где представлены некоторые параллельные алгоритмы для задач дискретной математики. В то время как в зарубежной периодике вопросам распараллеливания алгоритмов уделяется не сравнимо большее внимание.

Большинство известных параллельных графовых алгоритмов построено для PRAM (Parallel Random Access Machine) – модели параллельной вычислительной системы. В представленной раннее классификации, PRAM относится к компьютерам с общей памятью. Во избежание конфликтных ситуаций при од-

новременном обращении нескольких процессоров к общей памяти в PRAM заданы следующие правила:

- конкурентное чтение / конкурентная запись (CRCW);
- конкурентное чтение / исключительная запись (CREW);
- исключительное чтение / конкурентная запись (ERCW);
- исключительное чтение / исключительная запись (EREW).

С более подробным описанием PRAM модели можно ознакомиться, например, в [8] или [9].

Простейшие параллельные алгоритмы для графа G с n вершинами и m ребрами можно найти в переводном учебнике [10].

В [11] предлагается эффективный алгоритм нахождения связных компонент неориентированного графа G. В этом алгоритме используется $n+m$ процессоров CREW PRAM, а сам алгоритм исполняется за время $O(\log^{3/2} n)$. В [12] разработан параллельный алгоритм поиска всех пар кратчайших путей, где на модели CRCW PRAM достигается временная сложность $O(n^3/p+\log n)$, используя при этом p процессоров. Для проверки того является ли данное остовное дерево минимальным, разработан параллельный алгоритм [13] на EREW PRAM модели. Алгоритм исполняется за время $O(\log n)$ и использует $O((m+n)/\log n)$ процессоров. Для графа, степень каждой вершины которого $\Delta > 2$, существует алгоритм [14] вершинной раскраски в Δ цветов на EREW PRAM (допуская при этом, что граф не содержит $(\Delta-1)$-клик). Время выполнения алгоритма на $n/\log n$ процессорах равно $O(\log n)$. Для обновления остовного дерева минимального веса, при одновременном добавлении к исходному графу k новых вершин, используется алгоритм [15], время выполнения которого $O(\log k \cdot \log n)$. Алгоритм реализуется на EREW PRAM с $(k \cdot n)/(\log k \cdot \log n)$ процессорами. В [16] описан параллельный алгоритм поиска максимальных ациклических множеств. Алгоритм исполняется за $O(\sqrt{n}\log^3 n)$ время на n^2 процессорах EREW PRAM. В работе [17] предлагается эффективное выполнение алгоритма Эдмондса для нахождения оптимального ветвления графа на STAR машине. Время исполнения алгоритма – $O(n \cdot \log n)$, а число используемых процессоров – m. STAR машина основана на модели параллельной машины типа SIMD с вертикальной обработкой информации [5].

2. Фрактальные и предфрактальные графы

Термином *затравка* условимся называть какой-либо связный граф $H=(W,Q)$. Для определения *фрактального (предфрактального) графа* [18] нам потребуется операция *замены вершины затравкой (ЗВЗ)*. Суть операции ЗВЗ заключается в следующем. В данном графе $G=(V,E)$ у намеченной для замещения вершины $\tilde{v} \in V$ выделяется множество $\tilde{V}=\{\tilde{v}_j\} \subseteq V$, $j=1,2,...,\left|\tilde{V}\right|$, смежных ей вершин. Далее из графа G удаляется вершина $\tilde{v}$ и все инцидент-

ные ей ребра. Затем каждая вершина $\tilde{v}_j \in \tilde{V}$, $j = 1,2,..., \left|\tilde{V}\right|$, соединяется ребром с одной из вершин затравки $H = (W, Q)$. Вершины соединяются произвольно (случайным образом) или по определенному правилу, при необходимости.

Предфрактальный граф будем обозначать через $G_L = (V_L, E_L)$, где V_L – множество вершин графа, а E_L – множество его ребер. Определим его рекуррентно, поэтапно, заменяя каждый раз в построенном на предыдущем этапе $l = 1,2,..., L-1$ графе $G_l = (V_l, E_l)$ каждую его вершину затравкой $H = (W, Q)$. На этапе $l = 1$ предфрактальному графу соответствует затравка $G_1 = H$. Об описанном процессе говорят, что *предфрактальный граф* $G_L = (V_L, E_L)$ *порожден затравкой* $H = (W, Q)$. Процесс порождения предфрактального графа G_L, по существу, есть процесс построения последовательности предфрактальных графов $G_1, G_2, ..., G_l, ..., G_L$, называемой *траекторией*. Фрактальный граф $G = (V, E)$, порожденный затравкой $H = (W, Q)$, определяется бесконечной траекторией.

Использование операции ЗВЗ в процессе порождения предфрактального графа G_L, для элементов $G_l = (V_l, E_l)$, $l \in \{1,2,..., L-1\}$, его траектории позволяет ввести отображение $\varphi : V_l \to V_{l+1}$ или $\varphi(V_l) = V_{l+1}$, а в общем виде

$$\varphi^t(V_l) = V_{l+t}, \quad t = 1,2,..., L-l. \qquad (1)$$

В выражении (1) множество V_{l+t} – *образ* множества V_l, а множество V_l – *прообраз* множества V_{l+t}.

Для предфрактального графа G_L, ребра, появившиеся на l-ом, $l \in \{1,2,..., L\}$, этапе порождения, будем называть *ребрами ранга* l. *Новыми* ребрами предфрактального графа G_L назовем ребра ранга L, а все остальные ребра назовем – *старыми*.

Если из предфрактального графа G_L, порожденного n-вершинной затравкой H, последовательно удалить все старые ребра (ребра ранга l, $l = 1,2,..., L-1$), то исходный граф распадется на множество связных компонент $\{B_L^{(1)}\}$, каждая из которых изоморфна [19] затравке H. Множество компонент $\{B_L^{(1)}\}$ будем называть *блоками первого ранга*. Аналогично, при удалении из предфрактального графа G_L всех старых ребер рангов $l = 1,2,..., L-2$, получим множество *блоков* $\{B_L^{(2)}\}$ *второго ранга*. Обобщая, скажем, что при удалении из предфрактального графа G_L всех ребер рангов $l = 1,2,..., L-r$, получим множество $\{B_{L,i}^{(r)}\}$, $r \in \{1,2,..., L-1\}$, *блоком* r*-го ранга*, где $i = 1,2,..., n^{L-r}$ - порядковый номер блока. Блоки $B_L^{(1)} \subseteq G_L$ первого ранга также будем называть *подграф-затравками* H предфрактального графа G_L.Очевидно, что всякий блок $B_L^{(r)} = (U_L^{(r)}, M_L^{(r)})$, $r \in \{1,2,..., L-1\}$, является предфрактальным графом $B_r = (U_r, M_r)$, порожденным затравкой H.

Уточним для отображения φ в формуле (1) ряд подробностей. Для любой вершины $v_j \in V_l$, $j \in \{1,2,...,n^l\}$, предфрактального графа $G_l = (V_l, E_l)$, $l \in \{1,2,...,L-1\}$, из траектории графа G_L, справедливо

$$\varphi^t(v_j) = U_{l+t,j}^{(t)}, \qquad (2)$$

$$\varphi^t(v_j) = B_{l+t,j}^{(t)}, \text{ где } B_{l+t,j}^{(t)} = (U_{l+t,j}^{(t)}, M_{l+t,j}^{(t)}) \subseteq G_{l+t}, \ t = 1,2,...,L-l.$$

Аналогично,

$$\varphi^t(U_{l,i}^{(r)}) = U_{l+t,i}^{(r+t)}, \qquad (3)$$

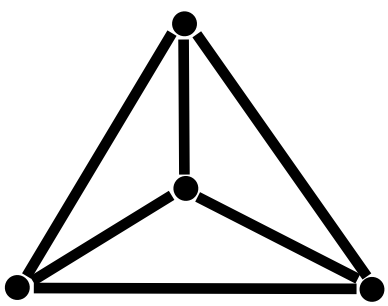
$$\varphi^t(B_{l,i}^{(r)}) = B_{l+t,i}^{(r+t)}, \ r \in \{1,2,...,L-t\}, \ i \in \{1,2,...,n^{l-r}\}.$$

Два блока предфрактального графа назовем *смежными*, если существует ребро, вершины которого принадлежат различным блокам. Не требует доказательства тот факт, что блоки предфрактального графа смежны тогда и только тогда, когда смежны их прообразы из (2).

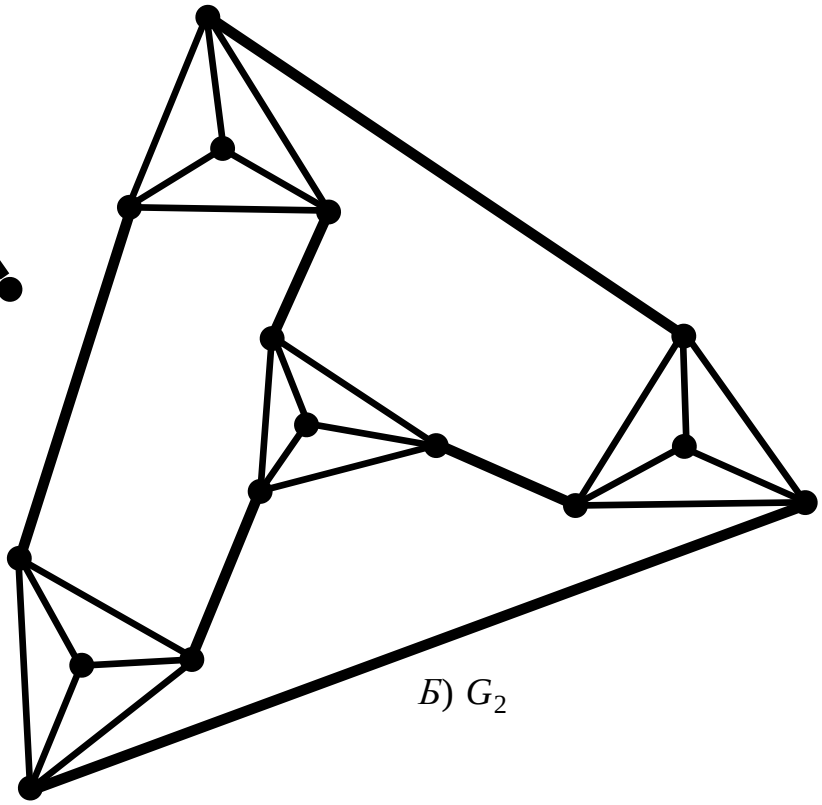
Обобщением описанного процесса порождения предфрактального графа G_L является такой случай, когда вместо единственной затравки H используется множество затравок $\mathbf{H} = \{H_t\} = \{H_1, H_2, ..., H_t, ..., H_T\}$, $T \geq 2$. Суть этого обобщения состоит в том, что при переходе от графа G_{l-1} к графу G_l каждая вершина замещается некоторой затравкой $H_t \in \mathbf{H}$, которая выбирается случайно или согласно определенному правилу, отражающему специфику моделируемого процесса или структуры.

Термином *подграф-затравка* $z_s^{(l)}$ будем называть блок $B_{l,s}^{(1)}$, $s = \overline{1, n^{l-1}}$, первого ранга предфрактального графа G_l, $l = \overline{1, L}$ из траектории. Последовательное выделение подграф-затравок $z_s^{(l)}$ на графах $G_1, G_2, ..., G_L$ из траектории предфрактального графа G_L разбивает множество ребер E_L на непересекающиеся подмножества подграф-затравок $Z(G_L) = \{z_s^{(l)}\}$, $l = \overline{1, L}$, $s = \overline{1, n^{l-1}}$. Такое разбиение на подмножества позволит нам сохранить информацию смежности старых ребер на момент их появления в предфрактальном графе. В траектории переход от графа G_{l-1} к G_l осуществляется $|V_{l-1}| = n^{l-1}$ операциями ЗВЗ, поэтому общее число использованных затравок в порождении предфрактального графа G_L равно $1 + n + n^2 + ... + n^{L-1} = \dfrac{n^L - 1}{n - 1}$. Тогда мощность множества $Z(G_L)$ всех подграф-затравок из траектории графа G_L также равно $Z(G_L) = \dfrac{n^L - 1}{n - 1}$.

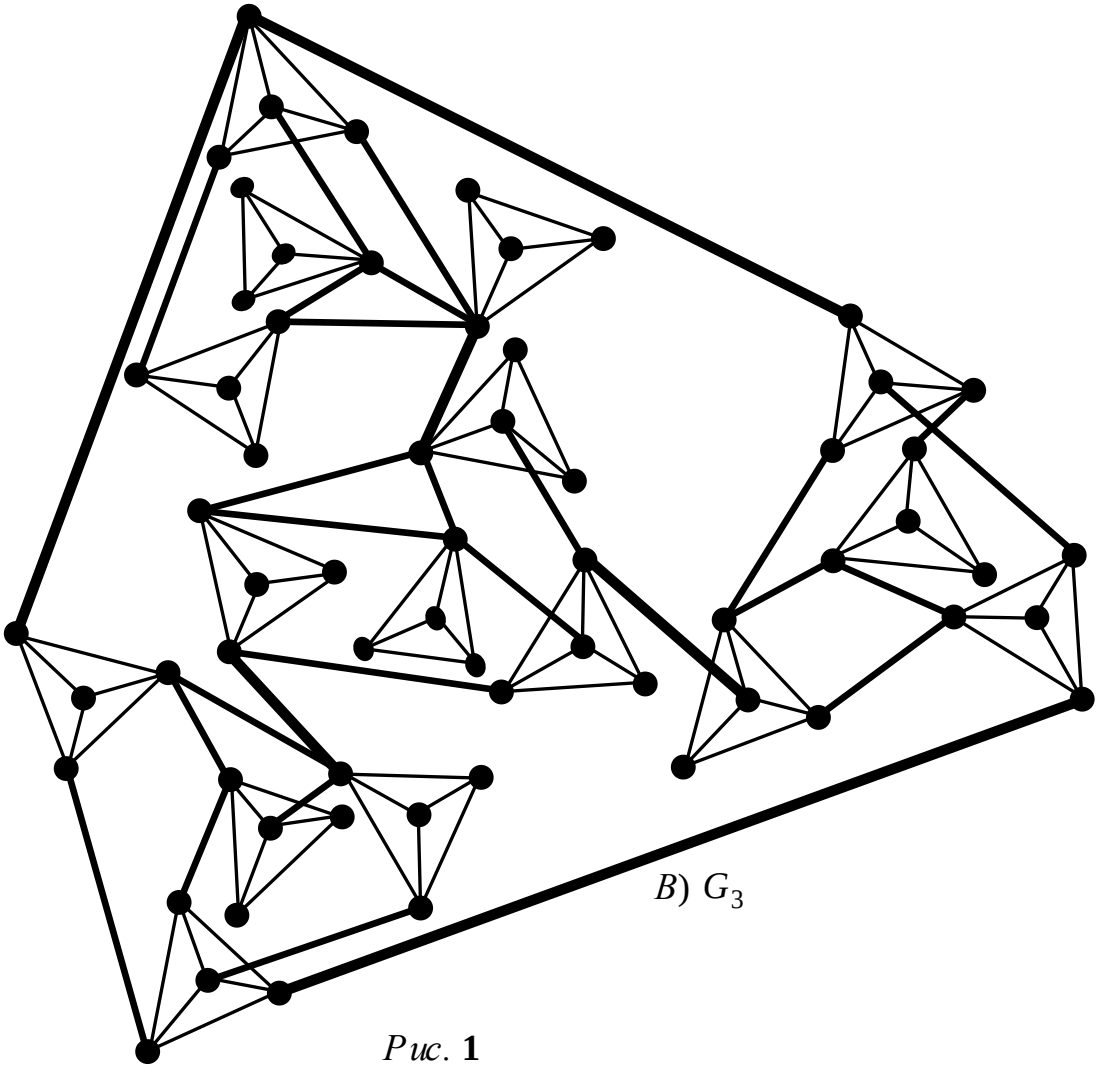
На рис. 1 изображена траектория предфрактального графа $G_3 = (V_3, E_3)$, порожденного затравкой $H = (W, Q)$ – полным 4-вершинным графом (см. рис. 1 *А*). Самыми “жирными” линиями нарисованы ребра подграф-затравки $z_1^{(1)}$.



А) $H = G_1$



Б) G_2



В) G_3

Рис. **1**

Линиями средней “жирности” нарисованы ребра подграф-затравок $z_1^{(2)}$, $z_2^{(2)}$, $z_3^{(2)}$ и $z_4^{(2)}$. И наконец, тонкими линиями (см. рис. 1 *B*) нарисованы новые ребра предфрактального графа G_3, которые образуют подграф-затравки $z_s^{(3)}$, $s=\overline{1,16}$.

Будем говорить, что *предфрактальный граф* $G_L=(V_L,E_L)$ *– взвешен*, если каждому его ребру $e^{(l)}\in E_L$ приписано действительное число $w(e^{(l)})\in(\theta^{l-1}a,\theta^{l-1}b)$, где $l=\overline{1,L}$ – ранг ребра, $a>0$, и $\theta<\dfrac{a}{b}$.

3. Параллельный алгоритм поиска ОДМВ

Параллельный алгоритм α_{Pr} осуществляет поиск остовного дерева минимального веса (ОДМВ) [19] $T=(V_L,E_T)$ на взвешенном предфрактальном графе $G_L=(V_L,E_L)$. Алгоритм использует *k* процессоров $p_1,p_2,...,p_k$. Назначим каждый процессор одной из подграф-затравок $z_s^{(l)}$, $l=\overline{1,L}$, $s=\overline{1,n^{l-1}}$, предфрактального графа G_L, тогда число используемых процессоров равно $k=\dfrac{n^L-1}{n-1}$.

Суть работы алгоритма заключается в следующем. Каждая подграф-затравка $z_s^{(l)}$ рассматривается как отдельно взятый граф. При этом каждый из *k* процессоров параллельно независимо друг от друга находит ОДМВ на своей подграф-затравке $z_s^{(l)}$. Поиск ОДМВ отдельно взятой подграф-затравки осуществляется алгоритмом Прима [20]. Алгоритм Прима используется в алгоритме α_{Pr} в виде процедуры, по мере необходимости. Нахождения ОДМВ всех подграф-затравок $z_s^{(l)}$, позволяет построить ОДМВ предфрактального графа G_L.

Каждое ребро предфрактального графа имеет свой “уникальный” номер, однозначно определяющий ребро во всей траектории. Таким образом, выделение ОДМВ на подграф-затравке $z_s^{(l)}$ будет соответствует выделению множества ребер на предфрактальном графе G_L.

Использование $k=\dfrac{n^L-1}{n-1}$ процессоров, равное числу подграф-затравок позволяет достичь, как это будет показано далее, максимальной эффективности алгоритма α_{Pr}.

Алгоритм α_{Pr}.

Вход: взвешенный предфрактальный граф $G_L=(V_L,E_L)$.

Выход: ОДМВ $T=(V_L,E_T)$.

1. Назначим каждый из *k* процессоров $p_1,p_2,...,p_k$ подграф-затравкам

$z_s^{(l)}$, $l=\overline{1,L}$, $s=\overline{1,n^{l-1}}$. Каждый процессор будет обрабатывать только назначенную ему подграф-затравку.

2. Одновременно k процессоров $p_1, p_2, ..., p_k$ параллельно и независимо друг от друга применяют процедуру Prim к назначенной подграф-затравке.

3. На выходе шага 2 получаем множество из k ОДМВ $T_1, T_2, ..., T_k$, которое определяет ОДМВ $T=(V_L, E_T)$, $E_T \subseteq E$.

ПРОЦЕДУРА Prim.

Вход: взвешенный граф $G=(V,E)$.

Выход: ОДМВ $T_s=(V,E_s)$.

Для обоснования алгоритма α_{Pr} будем использовать

УТВЕРЖДЕНИЕ 1. Всякий предфрактальный граф G_L можно представить в виде множества подграф-затравок $\{B_L^{(1)}\}$, соединенных старыми ребрами разных рангов. А именно, старые ребра $(L-1)$-го ранга объединяют множество подграф-затравок в множество блоков $\{B_L^{(2)}\}$ второго ранга, их в свою очередь, старые ребра $(L-2)$-го ранга объединяют в множество блоков $\{B_L^{(3)}\}$ третьего ранга и т.д. Окончательно, старые ребра первого ранга объединяют множество $\{B_L^{(L-1)}\}$ блоков $(L-1)$-го ранга в связный предфрактальный граф G_L.

ТЕОРЕМА 1. Для предфрактального графа $G_L=(V_L,E_L)$ всякий его связный остовный подграф $S=(V_L,E_S)$ удовлетворяет двум условиям:

1. В множество ребер E_S входит хотя бы одно ребро каждого ранга.

2. Старые ребра $\{e^{(l)}\}\subseteq E_S$ ранга l, $l\in\{1,2,...,L-1\}$, имеющие общую вершину-прообраз, на графе G_l из траектории, образуют связный подграф.

ДОКАЗАТЕЛЬСТВО. Первая часть теоремы целиком вытекает из утверждения 1. Действительно, если на подграфе $S=(V_L,E_S)$ будут отсутствовать все ребра какого-либо ранга, то он не будет связным. Вторую часть теоремы докажем от противного. Предположим, что некоторое множество ребер $\{e^{(l)}\}\subseteq E_S$, имеющих общую вершину-прообраз не образует на предфрактальном графе G_l, $l\in\{1,2,...,L-1\}$, связный подграф. Это значит, что среди образов концов ребер $\{e^{(l)}\}$ на графе G_L найдутся такие, что ни одна из цепей их соединяющих не войдет в остовый подграф $S=(V_L,E_S)$, что противоречит условию связности этого подграфа. ◀[1]

ТЕОРЕМА 2. Параллельный алгоритм α_{Pr} строит на предфрактальном графе $G_L=(V_L,E_L)$ ОДМВ $T=(V_L,E_T)$.

ДОКАЗАТЕЛЬСТВО. Алгоритм α_{Pr} на предфрактальном графе G_L выделяет k ОДМВ $T_1, T_2, ..., T_k$. Докажем, что множество $T_1, T_2, ..., T_k$ образует ОДМВ

[1] Здесь и далее символом "◀" будем обозначать конец доказательств лемм и теорем.

предфрактального графа G_L. ОДМВ T_i, $i=\overline{1,n^{L-1}}$, выделенные на подграф-затравках $z_s^{(L)}$, образуют остовный лес, состоящий из n^{L-1} связных компонент. Связные компоненты, в данном случае, это ОДМВ выделенные на блоках $\{B_L^{(1)}\}$. Согласно правилу взвешивания предфрактального графа, ребра принадлежащие блокам первого ранга имеют наименьшие веса по сравнению с ребрами других рангов. Поэтому выделение ОДМВ на подграф-затравках $z_s^{(L)}$ необходимо для получения ОДМВ предфрактального графа G_L.

Далее ОДМВ T_i выделенные на подграф-затравках $z_s^{(L-1)}$, в соответствии с утверждением 1, образуют n^{L-2} связных компонент. Каждая компонента будет представлять собой ОДМВ блоков $\{B_L^{(2)}\}$. Продолжая данную линию рассуждений получим, что ОДМВ выделенные на подграф-затравках $z_s^{(2)}$ в совокупности с раннее найденными ОДМВ образуют n связных компонент. Каждая компонента будет представлять ОДМВ блока $\{B_L^{(L-1)}\}$. И, наконец, ОДМВ подграф-затравки $z_1^{(1)}$ связывает n компонент в одну связную компоненту. Полученный связный остовный подграф, в силу добавления деревьев минимального веса, будет представлять ОДМВ предфрактального графа G_L. ◄

ТЕОРЕМА 3. Время исполнения алгоритма $\alpha_{\text{Рг}}$ на предфрактальном графе $G_L=(V_L,E_L)$, $|V_L|=n^L$ при использовании $k=\dfrac{n^L-1}{n-1}$ процессоров, равно $O(n^2)$.

ДОКАЗАТЕЛЬСТВО. Основой алгоритма является шаг 2, с ним и связано затрата времени на исполнение. Остальные шаги производят простые операции и инициализацию переменных. Алгоритм $\alpha_{\text{Рг}}$ выполняет шаг 2 параллельно, каждым из k процессоров. Для выполнения шага 2 потребуется $O(n^2)$ времени (столько времени [20] требует процедура Prim для обработки подграф-затравки). Так как это верхняя оценка в наихудшем случае, тогда будем считать что все k процессоров закончат свою работу – выполнение шага 2 – одновременно, за время $O(n^2)$. Тогда алгоритм $\alpha_{\text{Рг}}$ исполняется за время $O(n^2)$. ◄

Отметим, что n определяет размерность задачи поиска ОДМВ на подграф-затравке, в то время как размерность задачи поиска ОДМВ на предфрактальном графе $G_L=(V_L,E_L)$ равна N, $|V_L|=N=n^L$. Таким образом время исполнения алгоритма $\alpha_{\text{Рг}}$ равно времени затрачиваемому на поиск ОДМВ на подграф-затравке.

Время исполнения последовательного алгоритма Прима на предфрактальном графе $G_L=(V_L,E_L)$, $|V_L|=N$ равно $O(N^2)$. Сравнив последовательный алгоритм Прима с параллельным алгоритмом $\alpha_{\text{Рг}}$, получаем что время исполнения алгоритма $\alpha_{\text{Рг}}$ меньше на порядок: $O(N)<O(N^2)$.

СЛЕДСТВИЕ 3.1. Ускорение параллельного алгоритма α_{Pr} на предфрактальном графе $G_L = (V_L, E_L)$, $|V_L| = n^L$ при использовании k процессоров, $k = \dfrac{n^L - 1}{n-1}$, от последовательного алгоритма Прима равно $O(N)$.

ТЕОРЕМА 4. Вычислительная сложность алгоритма α_{Pr} на предфрактальном графе $G_L = (V_L, E_L)$, $|V_L| = N = n^L$ равна $O(N \cdot n^2)$.

ДОКАЗАТЕЛЬСТВО. Алгоритм α_{Pr} представляет собой, по существу многократное выполнение шага 2. Шаг 2 потребует выполнения $O(n^2)$ операций на каждой подграф-затравке (столько операций требует процедура Прима). В сумме будет выполнено $k \cdot O(n^2)$ операций, $k = \dfrac{n^L - 1}{n-1}$. ◄

Тогда, $O(k \cdot n^2) = O\left(\dfrac{n^L - 1}{n-1} \cdot n^2\right) = O(n^L \cdot n^2) = O(N \cdot n^2)$. Отсюда вычислительная сложность алгоритма α_{Pr} равна $O(N \cdot n^2)$.

Вычислительная сложность алгоритма Прима равна $O(N^2)$. Сравнив ее с вычислительной сложностью алгоритма α_{Pr}, получаем: $O(N \cdot n^2) < O(N^2)$.

СЛЕДСТВИЕ 4.1. Вычислительная сложность алгоритма α_{Pr} меньше вычислительной сложности алгоритма Прима в n^{L-2} раз.

4. ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОИСКА СОВЕРШЕННОГО ПАРОСОЧЕТАНИЯ

Алгоритм β_{Ed} производит поиск совершенного паросочетания [19] на заданном предфрактальном графе. Рассмотрим взвешенный предфрактальный граф $G_L = (V_L, E_L)$ и траекторию $G_l = (V_l, E_l)$, $l = 1,2,..,L$. Алгоритм использует k процессоров $p_1, p_2,..., p_k$. Где k может варьировать в промежутке от одного до n^{L-1}. Назначим каждый процессор одной из подграф-затравок $z_s^{(L)}$, $s = \overline{1, n^{L-1}}$.

ТЕОРЕМА 5. Предфрактальный граф $G_L = (V_L, E_L)$, порожденный затравкой $H = (W, Q)$, имеет совершенное паросочетание, если затравка $H = (W, Q)$ имеет совершенное паросочетание.

Основная идея алгоритма заключается в том, что каждая подграф-затравка рассматривается как отдельно взятый граф. При этом каждый из k процессоров параллельно независимо друг от друга находит совершенные паросочетания на своей подграф-затравке $z_s^{(L)}$. Поиск совершенного паросочетания подграф-затравки осуществляется с помощью алгоритма Эдмондса [19]. Алгоритм Эдмондса оформлен в виде процедуры и используется по мере необходимости. В результате нахождения совершенных паросочетаний всех под-

граф-затравок $z_s^{(L)}$, получаем совершенное паросочетание предфрактального графа G_L. ◄

Для достижения максимальной эффективности алгоритма β_{Ed} предположим, что число процессоров равно количеству подграф-затравок $z_s^{(L)}$, то есть $k = n^{L-1}$.

АЛГОРИТМ β_{Ed}.

ВХОД: предфрактальный граф $G_L = (V_L, E_L)$.

ВЫХОД: совершенное паросочетание $M = (V_L, E_M)$.

1. Назначим каждый из k процессоров $p_1, p_2, ..., p_k$ подграф-затравкам $z_s^{(L)}$, $s = \overline{1, n^{L-1}}$. Каждый процессор будет обрабатывать только назначенную ему подграф-затравку.

2. Одновременно k процессоров $p_1, p_2, ..., p_k$ параллельно и независимо друг от друга применяют процедуру Edmonds к назначенной подграф-затравке.

3. На выходе шага 2 получаем множество k совершенных паросочетаний $M_1, M_2, ..., M_k$, которое определяет совершенное паросочетание $M = (V_L, E_M)$, $E_M \subseteq E$.

ПРОЦЕДУРА Edmonds.

ВХОД: граф $G = (V, E)$.

ВЫХОД: совершенное паросочетание $M_s = (V, E_s)$.

ТЕОРЕМА 6. Параллельный алгоритм β_{Ed} строит на предфрактальном графе $G_L = (V_L, E_L)$ совершенное паросочетание $M = (V_L, E_M)$.

ДОКАЗАТЕЛЬСТВО. На последнем шаге L порождения предфрактального графа G_L все вершины замещаются затравкой $H = (W, Q)$. Совершенное паросочетание M_s подграф-затравки $z_s^{(L)}$, $s = \overline{1, n^{L-1}}$, покроет все вершины принадлежащие данной подграф-затравке. Так как множество вершин графа G_L можно представить в виде совокупности вершин подграф-затравок $z_s^{(L)}$, то покрыв совершенными паросочетаниями все подграф-затравки, получим покрытие всего множества вершин G_L. Совершенные паросочетания подграф-затравок $z_s^{(L)}$, при этом состоят только из новых ребер и в совокупности образуют совершенное паросочетание предфрактального графа G_L. ◄

ТЕОРЕМА 7. Время исполнения алгоритма β_{Ed} на предфрактальном графе $G_L = (V_L, E_L)$, $|V_L| = n^L$ при использовании $k = n^{L-1}$ процессоров, равно $O(n^3)$.

ДОКАЗАТЕЛЬСТВО. Основой алгоритма является шаг 2, с ним и связано затрата времени на исполнение. Остальные шаги производят простые операции и инициализацию переменных. Алгоритм β_{Ed} выполняет шаг 2 параллельно, каждым из k процессоров. Для выполнения шага 2 потребуется $O(n^3)$ времени

(столько времени [20] требует процедура Edmonds для обработки подграф-затравки). Так как это верхняя оценка в наихудшем случае, тогда будем считать что все k процессоров закончат свою работу – выполнение шага 2 – одновременно, за время $O(n^3)$. Тогда алгоритм β_{Ed} исполняется за время $O(n^3)$. ◄

Время исполнения последовательного алгоритма Эдмондса на предфрактальном графе $G_L = (V_L, E_L)$, $|V_L| = N$ равно $O(N^3)$. Сравнив последовательный алгоритм Эдмондса с параллельным алгоритмом β_{Ed}, получаем что время исполнения алгоритма β_{Ed} меньше на два порядка: $O(N) < O(N^3)$.

СЛЕДСТВИЕ 7.1. Ускорение параллельного алгоритма β_{Ed} на предфрактальном графе $G_L = (V_L, E_L)$, $|V_L| = n^L$ при использовании k процессоров, $k = n^{L-1}$, от последовательного алгоритма Эдмондса равно $O(N^2)$.

ТЕОРЕМА 8. Вычислительная сложность алгоритма β_{Ed} на предфрактальном графе $G_L = (V_L, E_L)$, $|V_L| = N = n^L$ равна $O(N \cdot n^2)$.

ДОКАЗАТЕЛЬСТВО. Алгоритм β_{Ed} представляет собой, по существу многократное выполнение шага 2. Шаг 2 потребует выполнения $O(n^3)$ операций на каждой подграф-затравке (столько операций требует процедура Edmonds). В сумме будет выполнено $k \cdot O(n^3)$ операций, $k = n^{L-1}$. ◄

Тогда, $O(k \cdot n^3) = O(n^{L-1} \cdot n^3) = O(n^L \cdot n^2) = O(N \cdot n^2)$. Отсюда, вычислительная сложность алгоритма β_{Ed} равна $O(N \cdot n^2)$.

Вычислительная сложность алгоритма Эдмондса равна $O(N^3)$. Сравнив ее с вычислительной сложностью алгоритма β_{Ed}, получаем: $O(N \cdot n^2) < O(N^3)$.

СЛЕДСТВИЕ 8.1. Вычислительная сложность алгоритма β_{Ed} меньше вычислительной сложности алгоритма Эдмондса в n^{2L-2} раз.

5. ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОИСКА ЭЙЛЕРОВОЙ ЦЕПИ

Для поиска эйлеровой цепи [19] предфрактального графа предложим алгоритм δ_{El}. Рассмотрим предфрактальный граф $G_L = (V_L, E_L)$ и его траекторию $G_l = (V_l, E_l)$, $l = 1,2,..,L$. Алгоритм использует k процессоров $p_1, p_2, ..., p_k$. Число процессоров не обязано соответствовать количеству входных записей, а может варьировать в промежутке от одного до $k = \dfrac{n^L - 1}{n - 1}$. Назначим каждый процессор одной из подграф-затравок $z_s^{(l)}$, $l = \overline{1, L}$, $s = \overline{1, n^{l-1}}$.

УТВЕРЖДЕНИЕ 2. Для того чтобы на предфрактальном графе G_L существовала эйлерова цепь, достаточно выполнение следующих условий:

1. Смежность старых ребер предфрактального графа G_L сохраняется.

2. На затравке $H = (W, Q)$ существует эйлерова цепь.

Таким образом, будем предполагать, что алгоритм требует выполнения условий утверждения 2.

Основная идея алгоритма заключается в следующем. Каждая подграф-затравка $z_s^{(l)}$ рассматривается как отдельно взятый граф. При этом каждый из k процессоров параллельно независимо друг от друга находит эйлерову цепь на своей подграф-затравке $z_s^{(l)}$. Поиск эйлеровой цепи отдельно взятой подграф-затравки осуществляется алгоритмом Эйлера [20]. Алгоритм Эйлера используется в алгоритме δ_{El} в виде процедуры по мере необходимости. В результате нахождения эйлеровых цепей всех подграф-затравок $z_s^{(l)}$, получаем эйлерову цепь предфрактального графа $G_L = (V_L, E_L)$.

Использование $k = \dfrac{n^L - 1}{n - 1}$ процессоров, равное числу подграф-затравок позволяет достичь максимальной эффективности алгоритма δ_{El}.

АЛГОРИТМ δ_{El}.

Вход: предфрактальный граф $G_L = (V_L, E_L)$.

Выход: эйлерова цепь $P = (V_P, E_L)$.

1. Назначим каждый из k процессоров $p_1, p_2, ..., p_k$ подграф-затравкам $z_s^{(l)}$, $l = \overline{1, L}$, $s = \overline{1, n^{l-1}}$. Каждый процессор будет обрабатывать только назначенную ему подграф-затравку.

2. Одновременно k процессоров $p_1, p_2, ..., p_k$ параллельно и независимо друг от друга применяют процедуру Eiler к назначенной подграф-затравке.

3. На выходе шага 2 получаем множество k эйлеровых цепей $P_1, P_2, ..., P_k$, которое определяет эйлерову цепь $P = (V_P, E_L)$.

ПРОЦЕДУРА Eiler.

Вход: граф $G = (V, E)$.

Выход: эйлерова цепь $P_s = (V_s, E)$.

ТЕОРЕМА 9. Параллельный алгоритм δ_{El} строит на предфрактальном графе $G_L = (V_L, E_L)$ эйлерову цепь $P = (V_P, E_L)$.

ДОКАЗАТЕЛЬСТВО. По определению, эйлерова цепь покрывает все ребра графа по одному разу, при этом любая вершина графа может состоять в цепи сколь угодно раз. Эйлерова цепь также представляет собой замкнутую цепь. На шаге 2 алгоритма δ_{El} найдены k эйлеровых цепей $P_1, P_2, ..., P_k$. Таким образом на подграф-затравке $z_1^{(1)}$ найдена эйлерова цепь P_1. Так как смежность старых ребер предфрактального графа G_L сохраняется, к эйлеровой цепи P_1 добавляются цепи P_i, выделенные на подграф-затравках $z_s^{(2)}$. В результате получаем эйлерову цепь G_2. Продолжая по данному принципу добавлять эйлеровы цепи

подграф-затравок $z_s^{(l)}$, в итоге получим эйлерову цепь предфрактального графа G_L. ◄

ТЕОРЕМА 10. Время исполнения алгоритма δ_{El} на предфрактальном графе $G_L = (V_L, E_L)$, $|E_L| = M = q\dfrac{n^L - 1}{n-1}$ при использовании $k = \dfrac{n^L - 1}{n-1}$ процессоров, равно $O(q)$.

ДОКАЗАТЕЛЬСТВО. Основой алгоритма является шаг 2, с ним и связано затрата времени на исполнение. Остальные шаги производят простые операции и инициализацию переменных. Алгоритм δ_{El} выполняет шаг 2 параллельно, каждым из k процессоров. Для выполнения шага 2 потребуется $O(q)$ времени (столько времени [20] требует процедура Eiler для обработки подграф-затравки). Так как это верхняя оценка в наихудшем случае, тогда будем считать что все k процессоров закончат свою работу – выполнение шага 2 – одновременно, за время $O(q)$. Тогда алгоритм δ_{El} исполняется за время $O(q)$. ◄

Отметим, что q определяет размерность задачи поиска эйлеровой цепи на подграф-затравке, в то время как размерность задачи поиска эйлеровой цепи на предфрактальном графе $G_L = (V_L, E_L)$ равна M, $|E_L| = M$. Таким образом время исполнения алгоритма δ_{El} равно времени затрачиваемому на поиск эйлеровой цепи на подграф-затравке.

Время исполнения последовательного алгоритма поиска эйлеровой цепи на предфрактальном графе $G_L = (V_L, E_L)$, $|E_L| = M$ равно $O(M)$. Сравнив последовательный алгоритм Эйлера с параллельным алгоритмом δ_{El}, получаем, что время исполнения алгоритма δ_{El} меньше на порядок: $O(q) < O(M)$.

СЛЕДСТВИЕ 10.1. Ускорение параллельного алгоритма δ_{El} на предфрактальном графе $G_L = (V_L, E_L)$, $|V_L| = N = n^L$ при использовании k процессоров, $k = \dfrac{n^L - 1}{n-1}$, от последовательного алгоритма Эйлера равно $O(N)$.

ТЕОРЕМА 11. Вычислительная сложность алгоритма δ_{El} на предфрактальном графе $G_L = (V_L, E_L)$, $|E_L| = M$ равна $O(M)$.

ДОКАЗАТЕЛЬСТВО. Алгоритм δ_{El} представляет собой, по существу многократное выполнение шага 2. Шаг 2 потребует выполнения $O(q)$ операций на каждой подграф-затравке (столько операций требует процедура Eiler). В сумме будет выполнено $k \cdot O(q)$ операций, $k = \dfrac{n^L - 1}{n-1}$. ◄

Тогда, $O(k \cdot q) = O\left(\dfrac{n^L - 1}{n-1} \cdot q\right) = O(M)$. Вычислительная сложность алгоритма δ_{El} равна $O(M)$.

СЛЕДСТВИЕ 11.1. Вычислительная сложность алгоритма δ_{El} равна вычислительной сложности алгоритма Эйлера.

6. Параллельный алгоритм поиска гамильтонова цикла

Алгоритм χ_{Et} производит поиск гамильтонова цикла [19] на заданном предфрактальном графе. Опишем принцип работы алгоритма χ_{Et}. Для этого рассмотрим предфрактальный граф $G_L = (V_L, E_L)$ и траекторию $G_l = (V_l, E_l)$, $l = 1,2,..,L$. Алгоритм использует k процессоров $p_1, p_2,..., p_k$. Где k может варьировать в промежутке от одного до n^{L-1}.

Утверждение 3. Для того чтобы на предфрактальном графе G_L существовал гамильтонов цикл, достаточно выполнение следующих условий:

1. Смежность старых ребер предфрактального графа G_L не сохраняется.

2. На затравке $H = (W, Q)$ существует гамильтонов цикл.

3. Между любой парой вершин затравки $H = (W, Q)$ существует гамильтонова цепь.

Таким образом, будем предполагать, что алгоритм требует выполнения условий утверждения 3.

Работа алгоритма начинается с того, что на подграф-затравке $z_1^{(1)}$ производится поиск гамильтонова цикла. Затем на каждой подграф-затравке $z_s^{(2)}$ выделяется гамильтонова цепь между двумя вершинами, которые являются концами ребер, выделенного на подграф-затравке $z_1^{(1)}$, гамильтонова цикла. Поиск гамильтоновых цепей на подграф-затравках $z_s^{(2)}$ осуществляется параллельно и независимо n процессорами. На следующем этапе n^2 процессоров также параллельно и независимо находят гамильтоновы цепи на подграф-затравках $z_s^{(3)}$. Поиск гамильтоновой цепи на подграф-затравке $z_s^{(3)}$ осуществляется между двумя вершинами, которые являются концами гамильтоновой цепи, найденной на предыдущем этапе. Работа алгоритма продолжается до тех пор, пока на этапе L не будут найдены гамильтоновы цепи подграф-затравок $z_s^{(L)}$. В результате нахождения гамильтоновых цепей всех подграф-затравок $z_s^{(l)}$, получаем гамильтонов цикл предфрактального графа G_L.

Основная идея алгоритма заключается в том, что каждая подграф-затравка рассматривается как отдельно взятый граф. При этом на $l = 1,2,..,L$ этапах каждый из n^{l-1} процессоров параллельно и независимо друг от друга находит гамильтонову цепь на своей подграф-затравке $z_s^{(l)}$. Поиск гамильтоновой цепи подграф-затравки осуществляется с помощью алгоритма Hamilton [20]. Алгоритм Hamilton оформлен в виде процедуры и используется по мере необходимости.

Использование $k = n^{L-1}$ процессоров, равное числу подграф-затравок $z_s^{(L)}$ позволяет достичь максимальной эффективности алгоритма χ_{Et}.

АЛГОРИТМ χ_{Et}.

Вход: предфрактальный граф $G_L = (V_L, E_L)$.

Выход: гамильтонов цикл $C = (V, E_C)$.

1. Процессор p_1 применяет процедуру Hamilton_Cycle к подграф-затравке $z_1^{(1)}$. На выходе процедуры получаем гамильтонов цикл C_1.

2. На данном шаге необходимо выполнить $(L-1)$ этапов для $l = 2,.., L$.

2.1. Назначим каждый из $r = n^{l-1}$ процессоров $p_1, p_2,..., p_r$ подграф-затравкам $z_s^{(l)}$, $s = \overline{1, n^{l-1}}$. Каждый процессор будет обрабатывать только назначенную ему подграф-затравку.

2.2. Одновременно $r = n^{l-1}$ процессоров $p_1, p_2,..., p_r$ параллельно и независимо друг от друга применяют процедуру Hamilton к назначенной подграф-затравке.

3. На выходе шага 2 получаем множество гамильтоновых цепей $C_2, C_3,..., C_s$, $s = (\frac{n^L - 1}{n-1} - 1)$, которое в совокупности с циклом C_1 определяет гамильтонов цикл $C = (V, E_C)$, $E_C \subseteq E$.

ПРОЦЕДУРА Hamilton_Cycle.

Вход: граф $G = (V, E)$.

Выход: гамильтонов цикл $C_s = (V, E_s)$.

ПРОЦЕДУРА Hamilton.

Вход: граф $G = (V, E)$.

Выход: гамильтонова цепь $C_s = (V, E_s)$.

ТЕОРЕМА 12. Параллельный алгоритм χ_{Et} строит на предфрактальном графе $G_L = (V_L, E_L)$ гамильтонов цикл $C = (V, E_C)$.

ДОКАЗАТЕЛЬСТВО. Гамильтонов цикл покрывает все вершины графа строго по одному разу, а покрытие ребер графа определяется вершинами цикла. На шаге 1 алгоритм χ_{Et} производит поиск гамильтонова цикла на подграф-затравке $z_1^{(1)}$. Затем осуществляется поиск гамильтоновых цепей на подграф-затравках $z_s^{(2)}$. Поиск проводится между двумя вершинами подграф-затравки $z_s^{(2)}$, которым инцидентны два старых ребра 1-го ранга, входящих в гамильтонов цикл подграф-затравки $z_1^{(1)}$. Более простыми словами, на подграф-затравке $z_1^{(1)}$ находится гамильтонов цикл, а затем вместо вершин вставляются гамильтоновы цепи соответствующих подграф-затравок $z_s^{(2)}$. Таким образом, получаем гамильтонов цикл предфрактального графа G_2. Продолжая по данному принципу добавлять гамильтоновы цепи подграф-затравок $z_s^{(l)}$, в итоге получим гамильтонов цикл предфрактального графа G_L. Так как покрыты все вер-

шины предфрактального графа G_L, тогда найденный гамильтонов цикл является искомым. ◄

Заключение

В выполненной работе можно выделить несколько важных аспектов.

Во-первых. Вычислительная сложность всех описанных алгоритмов решения теоретико-графовых задач на порядок ниже чем у известных аналогичных алгоритмов.

Во-вторых. Распараллеливание в исследованных задачах является геометрическим. Т.е. входные данные алгоритмов делятся на определенные группы, которые затем обрабатываются на различных процессорах. Входными данными, в нашем случае, являются предфрактальный граф, а группами, соответственно, являются все его подграф-затравки.

В-третьих. Это самый важный аспект. Возможность выделение на предфрактальном графе подграф-затравок, а значит и проведение геометрическое распараллеливания, существует только благодаря свойству самоподобия.

Считаем приятным долгом выразить признательность *профессору Малинецкому Г.Г.* за неоценимую помощь и поддержку в выполнении работы.

Литература

1. *Бандман О.Л.* Методы параллельного микропрограммирования. – Новосибирск: Наука, 1981.
2. *Горбунова Е.О.* Формально-кинетическая модель бесструктурного мелкозернистого параллелизма // Сибирский журнал вычислительной математики. 1999. Т. 2. № 3. С. 239-256.
3. *Бандман О.Л.* Мелкозернистый параллелизм в вычислительной математике // Программирование. 2001. №4. С. 5-20.
4. *Бочаров Н.В.* Технологии и техника параллельного программирования // Программирование. 2003. №1. С. 5-23.
5. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.
6. *Кочкаров А.А., Кочкаров Р.А*. Предфрактальные графы в проектировании и анализе сложных структур. Препринт ИПМ им. М.В. Келдыша РАН, №10, 2003.
7. *Кузюрин Н.Н.* Параллельный алгоритм для задачи о балансировке множеств. // Дискретная математика. 1991. Т.3. В.4. С. 153-158.
8. *Bader D.A., Illendula A.K., Moret B.M.E., Weisse-Bernstein N.R.* Using PRAM algorithms on a uniform-memory-access shared-memory architecture. WAE 2001. LNCS 2141. 2001. P. 129-144.
9. *Agbaria A., Ben-Asher Y., Newman I.* Communication–processor tradeoffs in a limited resources PRAM. Algorithmica. 2002. № 34. P. 276–297.
10. *Макконелл Дж.* Анализ алгоритмов. Вводный курс. – М.: Техносфера, 2002.

11. *Metaxas P.* Parallel Algorithms for Graph Problems. PhD dissertation, Dartmouth College, 1991.
12. *Han Y., Pan V.Y., Reif J.H.* Efficient parallel algorithms for computing all pair shortest paths in directed graphs. Algorithmica. 1997. № 17. P. 399–415.
13. *King V., Poon Ch.K., Ramachandran V., Sinha S.* An optimal EREW PRAM algorithm for minimum spanning tree verification. Information Processing Letters. 1997. № 62. P. 153-159.
14. *Sajith G., Saxena S.* Optimal parallel algorithm for Brook's colouring bounded degree graphs in logarithmic time on EREW PRAM. Discrete Applied Mathimatics. 1996. № 64. P. 249-265.
15. *Johnson D.B., Metaxas P.* Optimal algorithms for the single and multiple vertex updating problems of a minimum spanning tree. Algorithmica. 1996. № 16. P.633–648.
16. *Chen Z.-Z., He X.* Parallel algorithms for maximal acyclic sets. Algorithmica. 1997. № 19. P. 354-368.
17. *Непомнящая А.Ш.* Представление алгоритма Эдмондса для нахождения оптимального ветвления графа на ассоциативном параллельном процессоре // Программирование. 2001. №4. С. 43-52.
18. *Кочкаров А.М.* Распознавание фрактальных графов. Алгоритмический подход. – Нижний Архыз: РАН САО, 1998.
19. *Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И.* Лекции по теории графов. – М.: Наука, 1990.
20. *Асанов М.О., Баранский В.А., Расин В.В.* Дискретная математика: графы, матроиды, алгоритмы. – Ижевск: НИЦ "РХД", 2001.