РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОГО МНОГОУРОВНЕВОГО АЛГОРИТМА РАВНОМЕРНОГО РАЗБИЕНИЯ НЕРАСПРЕДЕЛЕННОГО ГРАФА

Н.В. Старостин, А.В. Филимонов

Нижегородский госуниверситет им. Н.И. Лобачевского

В работе формально поставлена задача равномерного k-разбиения графа, предложен адаптивный параллельный алгоритм решения задачи для случая нераспределенного графа, базирующийся на многоуровневой парадигме, представлены результаты вычислительных экспериментов для серии тестовых графов.

Введение

Численное решение задач математической физики на распределенных ВС предполагает использование распределенной между вычислительными узлами сетки, аппроксимирующей расчетную область. При этом каждый вычислительный узел обрабатывает данные на своей части сетки. Эффективность параллельных вычислений в значительной мере зависит от объёма межпроцессорных коммуникаций и сбалансированности загрузки вычислительных узлов. Выбор «хорошего» варианта декомпозиции сетки может привести к значительному ускорению параллельных расчетов на распределенных данных. Задача сбалансированного разбиения сетки по узлам вычислительной системы сводится к задаче сбалансированного k-разбиения графа, в которой требуется распределить вершины графа по заданному числу подграфов. Критерий задачи направлен на минимизацию внешних связей (коммуникация между вычислительными узлами). Под сбалансированностью понимаются по возможности равные веса подграфов разбиения (загрузка вычислительных узлов).

Задача k-разбиения графа

Рассматривается взвешенный неориентированный граф $G(V,E,w,u)$, где $V=\{v_1,...,v_n\}$ – множество вершин; $E\subseteq V^{(2)}$ – множество ребер; $w:V\to N$ отображение, приписывающее натуральный «вес» каждой вершине; $u:E\to N$ отображение, указывающее натуральный «вес» каждому ребру графа. Задача *k*-разбиения графа $G(V,E,w,u)$ состоит в распределении всех вершин из множества V по k подмножествам $V_1,...,V_k$, таким образом, чтобы удовлетворялось ограничение (1), а оптимизируемая функция (2) принимала экстремальное значение

$$\max_{i=1,k} W(V_i) \le c\frac{W(V)}{k},$$

где $W(S)=\sum_{v\in S} w(v)$, $S\subseteq V$, (1)

$c\in[1,\infty)$ – параметр разбиения.

Часто в задаче разбиения графа вместо параметра c используют параметр $\varepsilon = 100\cdot(c-1)$, который называют дисбалансом (с англ., imbalance).

$$F(V_1,...,V_k) = \sum_{e \in Q(V_1,...,V_k)} u(e) \to \min, \tag{2}$$

где $Q(V_1,...,V_k) = \left\{ \{v_i, v_j\} \in E \mid v_i \in V_l,\ v_j \in V_j, i = \overline{1,k}, j = \overline{1,k}, i \neq j \right\}$.

Задачу (1), (2) разбиения графа часто обозначают (k,c) задачей разбиения. Задачу $(k,1)$ называют задачей сбалансированного *k*-разбиения. В общем случае (k,c)-задача принадлежит к классу NP-трудных задач.

Общая схема многоуровневого алгоритма

Ключевая идея многоуровневого алгоритма (ML) разбиения состоит в следующем: вместо того, чтобы строить k-разбиение непосредственно для исходного графа, сначала строится ряд его приближений (загрублений). Каждое загрубление уменьшает (редуцирует) размерность исходной задачи. Процесс редукции продолжается до тех пор, пока порядок графа не снизится до сотен или даже десятков. На графе таких размерностей и отыскивается так называемое начальное разбиение. Полученное решение используется для построения решения для исходной задачи. Это достигается путем выполнения серии переносов решения задачи меньшей размерности на задачу большей размерности с последующим улучшением перенесенного решения. Таким образом, работу многоуровневого алгоритма можно разделить на три фазы: первая – фаза загрубления, когда производится ряд последовательных редукций размерности задачи, вторая – фаза поиска начального разбиения и третья – фаза восстановления решения, когда производится серия последовательных переносов решения задачи меньшей размерности на менее редуцированную задачу с последующим улучшением спроецированного решения. Работа алгоритма для задачи *k*-разбиения показана на рис.1.

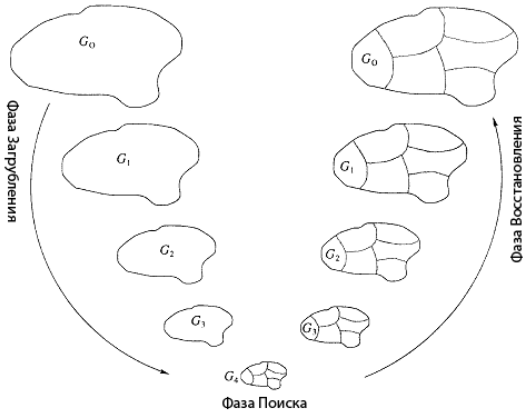

Рис. 1. Многоуровневый алгоритм 5-разбиения графа

Важным достоинством многоуровневых схем являются широкие возможности по их настройке под конкретные классы задач, так как они позволяют относительно просто интегрировать в свою структуру различные эффективные эвристические алгоритмы, направленные как на поиск сбалансированных разбиений, так и на улучшение получаемых решений. Подобные многоуровневые алгоритмы различаются: 1) стратегиями, используемыми для упрощения задачи и/или редукции графа; 2) алгоритмами, строящими начальное разбиение редуцированного графа; 3) эвристиками, используемыми для получения «удачной» проекции решения на этапе восстановления; 4) моделями, используемыми для настройки параметров многоуровневого алгоритма.

Структура многоуровневого итерационного алгоритма

По своей природе известные многоуровневые алгоритмы решения задачи (k,c)-разбиения ближе к конструктивным алгоритмам. На фазе загрубления используются стратегии, главная цель которых – получить относительно небольшой граф, который структурно «похож» на исходный граф. На фазе поиска используются конструктивные схемы генерации разбиений. Фаза восстановления использует улучшающие схемы, но они направлены исключительно на улучшение проекций начальных разбиений, полученных в фазе поиска.

Известные многоуровневые схемы позволяют «с нуля» строить новые решения. Эти алгоритмы можно использовать для генерации нескольких решений методом включения в общую схему элементов рандомизации либо изменением параметров алгоритма. Главный минус подобных схем в том, что они не обладают «обратной связью» – не используют информацию о предыдущих итерациях своей работы.

Предлагается многоуровневый итерационный алгоритм (MLI-алгоритм) задачи (k,c)-разбиения. Алгоритм на вход получает не только исходные данные задачи (G, k, c), но и одно из допустимых решений (k,c) задачи. Цель алгоритма – исследование областей вокруг начального решения и поиск лучшего решения.

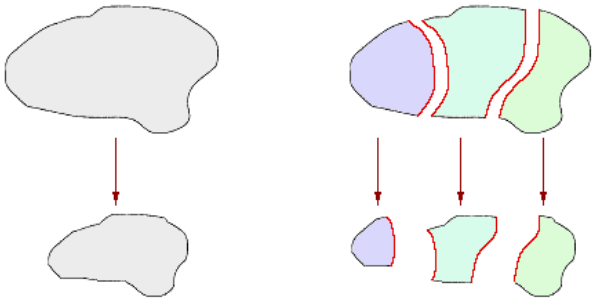

Рис. 2. Различия в концепциях загрубления классического (слева) и итерационного (справа) многоуровневых алгоритмов

Процесс работы MLI также включает три фазы. На первой фазе (фаза загрубления) алгоритм в отличии от MA осуществляет последовательную редукцию и графа (загрубление задачи), и начального разбиения (загрубление решения), если оно имеется. На рисунке 2 показаны отличия принципов работы фазы загрубления классического ML и MLI. Каждое загрубление уменьшает (редуцирует) размерность подграфа разбиения исходной задачи. При этом происходит уменьшение размерности исходного графа, а что самое важное – множество редуцированных подграфов разбиения определяет допустимое разбиение редуцированного графа.

Процесс редукции продолжается не более определенного числа раз (предельный уровень редукции). Процесс может прерываться в случае нехватки оперативной памяти, бесперспективности редукции или по ограничению MLI по времени работы.

Вторая фаза – фаза поиска решения самой загрубленной задачи. ML на этом этапе применяет две стратегии поиска лучшего решения. Первая стратегия заключается в локальном улучшении загрубленного начального решения. Вторая стратегия предполагает поиск нового решения загрубленной задачи. В завершении фазы поиска из двух найденных допустимых разбиений MLI выбирает лучшее с точки зрения критерия (2).

Третья фаза – фаза восстановления решения. Здесь MLI последовательно переходит от задачи более редуцированной к задаче менее редуцированной. При каждом переходе алгоритм восстанавливает и улучшает решение более редуцированной задачи. В этот момент MLI владеет парой решений менее редуцированной задачи: начальным загрубленным и улучшенным решением и найденным восстановленным и улучшенным реше-

нием. МА выбирает лучшее с точки зрения критерия (2) и переходит к следующей менее редуцированной задаче.

Описанная схема работает в итерационном режиме так: начальное решение улучшается MLI, найденное лучшее решение становится начальным для следующей итерации MLI. Итерационный процесс продолжается не более определенного числа раз (максимальное число итераций), и может прерываться в случае нехватки памяти, бесперспективности итерации или в связи с нехваткой времени на работу алгоритма.

В качестве алгоритма поиска начального решения в описанной схеме используется алгоритм рекурсивной бисекции, цель которого - из задачи k-разбиения графа получить серию более «простых» задач 2-разбиения графа, при этом в случае наличия начального решения исходной задачи для каждой задачи 2-разбиения графа формируется ее начальная бисекция. Решение каждой из задач бисекции происходит с помощью многоуровневого алгоритма бисекции

Анализ выполнения алгоритма показал, что наиболее ресурсоемкой процедурой, стоящей около 90% общего времени работы алгоритма, является именно бисекция. Поэтому наиболее эффективным с точки зрения сокращения времени работы алгоритма было бы построение параллельной реализации этой процедуры.

Параллельный вариант этапа рекурсивной бисекции

Декомпозиция рекурсии заключается в получении из одной задачи k-разбиения двух новых задач k_1-разбиения и k_2-разбиения, где $k = k_1 + k_2$, причем $|k_1 - k_2| \leq 1$. При этом из одной задачи две новые задачи получаются методом решения задачи несбалансированного 2-разбиения. Процесс многоуровневой рекурсии продолжается до тех пор, пока все задачи не будут решены, и как следствие, не будет решена задача k-разбиения. Получающиеся в процессе декомпозиции новые задачи можно решать независимо друг от друга. Верхняя оценка ускорения параллельного варианта алгоритма при независимом решении задач бисекции на всех уровнях рекурсии составит

$$S(k,p) = \frac{p\lfloor \log_2 k \rfloor}{2p - 2 + \lfloor \log_2 k \rfloor - \lceil \log_2 p \rceil}, \qquad (3)$$

здесь p – количество процессоров.

Вычислительный эксперимент

Испытывалась программная реализация предложенных методов решения задач k-декомпозиции под кодовым названием PARMATRUZ. Качество получаемых решений сравнивалось с качеством решений известной реализации параллельного многоуровневого алгоритма k-way в рамках библиотеки PARMETIS v.4.0.2.

Вычислительный эксперимент проводился на КС-ЭВМ 1,1 TFlops ОС Unix x64.

Выбиралась настройка параметров по умолчанию для программ PARMETIS и PARMATRUZ. Все решения отыскивались в пределах 5% дисбаланса.

Таблица 1. Сравнение параллельных версий программ на КС-ЭВМ

Граф	Вершины ребра	Подграфы	Ядер	PARMETIS	PARMATRUZ	
				Сечение	Сечение	Время (сек)
800835	526 941 15 78 501	32	1	11553	11552	20
			4	12627	11552	7
			8	12711	11552	6
800835		1024	1	85207	87052	164
			4	91827	87052	43
			8	92081	87052	25
trubka	2 428 323	32	1	1214582	1211023	509

	67 910 216		4	1231592	1211023	169
			8	1233087	1211023	129
trubka		1024	1	7145238	7232042	3360
			4	7161643	7232042	844
			8	7145126	7232042	508
grid_200x200x250	10^7 29 860 000	32	1	394805	400634	710
			4	418197	400634	245
			8	432961	400634	230
grid_200x200x250		1024	1	1589231	1681869	3881
			4	1801544	1681869	1103
			8	1784045	1681869	695

Выводы

По результатам проведенных экспериментов можно сформулировать выводы:

- MATRUZ продемонстрировал возможность находить решения, сравнимые по качеству, а в ряде случаев превосходящие, решения PARMETIS.
- MATRUZ способен решать задачи k-разбиения графов размером порядка 10^7 числом вершин и значением k порядка 10^4.
- Ускорение параллельной версии MATRUZ согласуется с теоретическими оценками.
- Число процессоров влияет на время работы алгоритма, но не оказывает влияния на качество работы MATRUZ.