ОПРЕДЕЛЕНИЕ РЕСУРСА ПАРАЛЛЕЛИЗМА АЛГОРИТМОВ НА БАЗЕ КОНЦЕПЦИИ Q-ДЕТЕРМИНАНТА

С.В. Игнатьев

Введение

В этой статье предлагается метод выявления доступного ресурса параллелизма, основанный на анализе самих алгоритмов, а не их конкретных реализаций.

В основе любой программы лежат некоторые алгоритмы, решающие определенную проблему, причем для решения одной и той же проблемы, может существовать несколько алгоритмов. Важным является тот факт, что от выбора конкретного алгоритма, зачастую зависит куда больше, чем от качества его программной реализации. Так, если будет взят *неподходящий* алгоритм, то полученная на его основе реализация, может не обладать достаточными возможностями для распараллеливания. Тогда как выбор алгоритма, обладающего высоким ресурсом параллелизма, напротив, позволит получить реализацию, предоставляющую богатые возможности по оптимизации. Таким образом, выбор алгоритма задает верхний предел возможности распараллеливания.

Для анализа алгоритмов можно использовать их *представление в форме Q-детерминанта* [1]. Представление алгоритма в форме *Q*-детерминанта позволяет рассчитать такие сложностные характеристики алгоритма, как число процессоров и число тактов работы *вычислительной системы*, требующихся при выполнении его *максимально быстрой реализации*. Таким образом, можно получить представление о потенциале распараллеливания любого алгоритма, представленного в форме *Q*-детерминанта. Однако на текущий момент не существует методики, позволяющей строить *Q*-детерминант для произвольного алгоритма, предполагается, что *если* алгоритм удается представить в форме *Q*-детерминанта, то он может быть подвергнут анализу. В соответствии с этим, *актуальной* является задача разработки эффективного метода построения *Q*-детерминанта алгоритма.

1. Понятие *Q*-детерминанта

В данной части приводится описание метода распараллеливания численных алгоритмов. С этой целью для некоторого множества базовых операций *Q* вводится понятие представления алгоритма в форме *Q*-детерминанта [3]. Для алгоритмов, представленных в форме *Q*-детерминанта, описывается максимально быстрая реализация. Кроме того, предлагаются методы оценки таких важных характеристик алгоритма, как время выполнения максимально быстрой реализации на параллельной вычислительной системе (ВС) и число занятых процессоров.

Приведем основные понятия, используемые в рамках рассматриваемой концепции [1].

Понятие выражения

Пусть $B=\{b_1,b_2,\ldots\}$ – счетное множество переменных арифметического или логического типа, *Q* – конечное множество *арифметических* операций (таких как сложение, вычитание и изменение знака, умножение, деление и вычисление абсолютной величины числа) и *операций сравнения* (меньше, больше, равно, не равно, меньше или равно, больше или равно, логическое отрицание, логическое умножение, логическое сложение). *Q* будем называть множеством базовых операций.

Определим понятие *выражения* над множествами *B* и *Q*:

Константа и b_i из *B* являются выражениями. Их будем называть выражениями нулевого уровня вложенности.

Выражение, заключенное в круглые скобки, является выражением. Заключение выражения в круглые скобки не меняет его уровня вложенности.

Применяя одноместную базовую операцию к выражению (*i*-1)-го уровня вложенности, получаем выражение *i*-го уровня вложенности, а исходное выражение называется его подвыражением (*i*-1)-го уровня вложенности. Саму базовую операцию будем называть в этом случае операцией *i*-го уровня вложенности.

Применяя двухместную базовую операцию к выражениям соответственно i_1 и i_2 уровней вложенности, получаем выражение *i*-го уровня вложенности, где $i=\max(i_1, i_2) + 1$. Исходные выражения называются его подвыражениями соответственно i_1 и i_2 уровней вложенности. Саму базовую операцию будем называть *i*-го уровня вложенности.

Других выражений нет.

Выражение, образованное в результате применения к *n* выражениям *n*-1 раз одной из ассоциативных операций, называется цепочкой данных выражений длины *n*.

Интерпретацией переменной b_i из *B* будем называть присвоение переменной b_i конкретного значения. Если задана интерпретация всех переменных, входящих в выражение, то будем говорить, что задана интерпретация выражения. Если задана некоторая интерпретация выражения, то его можно вычислить.

Если при вычислении выражения обнаруживается, что необходимо выполнить операцию деления на ноль, то будем считать, что значение выражения не определено.

Понятие *Q*-терма

Назовем любое выражение над B и Q *безусловным Q-термом* и будем обозначать его w. Далее под *вычислением безусловного Q-терма* w при интерпретации B будем понимать вычисление выражения w. *Уровнем вложенности* безусловного Q-терма w будем называть уровень вложенности выражения w и обозначать T^w.

Если выражение w при любой интерпретации B принимает значение логического типа, то безусловный Q-терм w будем называть *безусловным логическим Q-термом*.

Пусть $w_1,\dots w_l$ – безусловные Q-термы, $u_1,\dots,u_l$ – безусловные логические Q-термы. Множество l пар (u_i, w_i) $(i=1,\dots,l)$ будем называть *условным Q-термом длины l*. Опишем вычисление условного Q-терма при заданной интерпретации B: следует вычислить выражения u_i, w_i $(i=1,\dots,l)$. Если существуют выражения u_j, w_j $(j=1,\dots,s;\ s < l+1)$, такие, что значение u_j равно единице, а w_j определено, то будем считать, что исходный Q-терм принимает значения w_j. В противном случае считаем, что значение Q-терма при данной интерпретации B не определено.

Счетное множество пар безусловных Q-термов $\{(u_i, w_i)\}_{i=1,2,\dots}$ будем называть *условным бесконечным Q-термом*, если для любого L $\{(u_i, w_i)\}_{i=1,\dots,L}$ является условным Q-термом. Опишем *вычисление условного бесконечного Q-терма* при заданной интерпретации B: чтобы вычислить условный бесконечный Q-терм, необходимо, вычисляя выражения u_i, w_i $(i=1,2,\dots)$., найти u_{j0}, w_{j0}, такие, что значение u_{j0} равно единице, а значение w_{j0} определено. Тогда в качестве значения Q-терма нужно взять w_{j0}. Если установлено, что выражений u_{j0}, w_{j0} не существует, то значение Q-терма при данной интерпретации B не определено.

Понятие *Q*-детерминанта алгоритма

Пусть A – некоторый алгоритм для решения алгоритмической проблемы $y=F(B)$, где B – множество входных данных, $y=(y_1,\dots,y_m)$ – множество выходных данных. Пусть I_1, I_2, I_3 – множества индексов (одно или два из множеств могут быть пустыми).

Рассмотрим множество Q-термов, удовлетворяющее условиям:

f_{i1} - безусловный Q-терм, i_1 из I_1;

f_{i2} - условный Q-терм, i_2 из I_2;

f_{i3} - условный бесконечный Q-терм, i_3 из I_3.

Предположим, что алгоритм A состоит в том, что для определения y_i требуется вычислить Q-терм f_i. Тогда множество Q-термов f_i будем называть *Q-детерминантом алгоритма A*, а представление алгоритма A в виде $y_i=f_i$ – представлением в форме Q-детерминанта.

Анализ распараллеливания численных алгоритмов

Реализацией алгоритма A, представленного в форме Q-детерминанта $y_i=f_i$, будем называть вычисление Q-термов f_i.

Если реализация такова, что две или более базовых операции выполняются одновременно, то такую реализацию будем называть *параллельной*.

Если реализация алгоритма A производится с помощью ВС, то будем говорить, что реализация алгоритма A выполняется на ВС или алгоритм A реализуется на ВС.

Относительно ВС, на которой будут реализовываться алгоритмы, предположим, что:

1. число процессоров не ограничено;
2. процессоры идентичны, выполняют все базовые операции;
3. время выполнения процессорами базовых операций одинаково, это время будем называть тактом;
4. оперативная память однородна и неограниченна;
5. процессоры имеют прямой доступ к оперативной памяти и могут выбирать из нее одновременно одну и ту же информацию, причем время выборки единицы информации значительно меньше времени ее обработки на процессоре.

Опишем некоторую реализацию алгоритма A, представленного в форме Q-детерминанта.

Зададим интерпретацию переменных B. Выражения всех составляющих Q-термов будем вычислять одновременно. При этом, если встречаются одинаковые выражения и подвыражения, то не имеет смысла вычислять все, достаточно вычислить одно из них, т.е. выполнять вычисления без дублирования. Будем говорить, что базовая операция *готова к выполнению*, если определены значения ее операндов. При вычислении каждого из выражений будем выполнять операции тотчас же, как они готовы к выполнению. Если для некоторого выражения u_j получено значение ноль, то вычисление соответствующего ему выражения w_j прекращается. Если для некоторой пары выражений (u_i, w_i) при вычислении установлено, что значение одного из выражений не определено, то вычисление другого выражения прекращается. Если для некоторой пары выражений u_{j0}, w_{j0} составляющих условный бесконечный Q-терм в результате вычисления установлено, что их значения определены и значение u_{j0} равно единице, то вычисление остальных выражений данного условного бесконечного Q-терма прекращается.

Описанную реализацию алгоритма *A* будем называть *максимально быстрой*. Если максимально быстрая реализация является параллельной, то такую реализацию назовем *максимально параллельной*.

Будем говорить, что реализация алгоритма *A* *выполнима*, если при выполнении ее на ВС на каждом такте работы ВС требуется конечное число процессоров.

Для алгоритма *A*, представленного в форме *Q*-детерминанта, могут быть рассчитаны следующие характеристики:

D_A– *число тактов* работы ВС, требующихся при выполнении максимально быстрой реализации;

P_A – *число процессоров* ВС, требующихся при выполнении максимально быстрой реализации [1].

2. Метод построения *Q*-детерминанта

В данной части рассматривается способ записи произвольного алгоритма, предлагается альтернативная форма представления алгоритма, а также приводится метод преобразования алгоритма к форме *Q*-детерминанта.

2.1. Представление алгоритма

Из возможных форм записи алгоритмов выберем графическую с помощью блок-схем: описание алгоритма с помощью блок схем осуществляется рисованием последовательности геометрических фигур, каждая из которых подразумевает выполнение определенного действия алгоритма. Порядок выполнения действий указывается стрелками.

Из множества допустимых (написание алгоритмов с помощью блок-схем регламентируется ГОСТом [2]) элементов блок-схемы, ограничимся рассмотрением лишь 4-х:

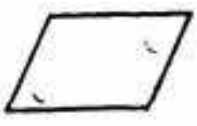

Рис. 1 Символ «Данные»

Данные. Символ отображает данные, носитель данных не определен.

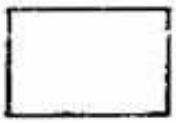

Рис. 2 Символ «Процесс»

Процесс. Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться).

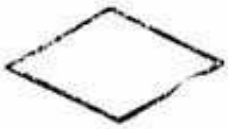

Рис. 3 Символ «Решение»

Решение. Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути.

Терминатор. Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).

Рис. 4 Символ «Терминатор»

Проведем некоторые уточнения и наложим дополнительные условия на алфавит используемых блок-схем.

Для удобства рассмотрения алгоритмов, положим, что блок-схема каждого алгоритма имеет ровно 2 терминирующих символа, один из которых обозначает начало алгоритма (далее по тексту *begin*), а другой – его окончание (*end*). Причем, для символа *begin* имеем отсутствие входящих связей и ровно 1 исходящую, а для *end* – отсутствие исходящих и не менее 1 входящей.

Не нарушая общности, будем считать, что из символа «Решение» (*condition*) имеется ровно 2 исходящие связи, одна из которых соответствует передачи потока управления, в случае если данное условие истинно (*true-ветвь*), а другая, если оно ложно (*false-ветвь*). Действительно, если у символа «Решение» более 2-х исходящих связей, он может быть заменен на последовательность элементов типа *condition* (по аналогии с заменой case-конструкций на последовательность вложенных конструкций if/else). Если же у него всего 1 исходящая связь – то, в силу определения алгоритма (из свойства детерминированности), можно удалить его из рассматриваемой блок-схемы.

Также, наложим ограничение на количество и характер действий внутри символа «Процесс» (*execution*). Далее считаем, что *execution* содержит не более 1-го действия и это действие – смена значения информации. Группы операций, приводящих к изменению значения информации, – легко заменяются на цепочки *execution*. Выполнение операций, приводящее к изменению формы или размещения информации считаем не существенными для алгоритма, а действия, приводящие к определению, по которому из нескольких направлений потока следует двигаться, реализуются с помощью *condition*.

Дополнительно, для уточнения направления потока данных, специфицируем символ «Данные»: если текущий символ подразумевает получение алгоритмом некоторых данных (например, задание конкретных условий решаемой алгоритмом задачи) – то назовем его *var_in*; если же подразумевается вывод информации из алгоритма (например, получение результата выполнения алгоритма) – *var_out*.

Особо отметим, что количество исходящих связей для всех элементов равно 1. Исключения составляют: *end*, у которого исходящие связи отсутствуют, и *condition* – ровно 2 исходящие связи. Количество входящих связей – не менее единицы, для всех элементов, кроме *begin*, входящие связи в который отсутствуют.

Подобные ограничения обусловлены свойствами алгоритмов (детерминированностью, понятностью, конечностью записи и дискретностью).

Очевидно, что выбранных элементов с установленными на них ограничениями, вполне достаточно, чтобы записать любой алгоритм.

2.2. Общая идея метода

Для построения *Q*-детерминанта алгоритма воспользуемся обратным проходом блок-схемы данного алгоритма с сохранением выполняемых операций и пройденных условий. Особым образом будем обрабатывать ситуации, когда в некоторый элемент блок схемы передают управление 2 и более элемента. Подобные «двоения» обнаруживаются, если блок-схема алгоритма содержит:

- простые условия;
- простые циклы;
- вложенные циклы;
- взаимовложенные (равноправные) циклы.

Рассмотрим характерные черты и особенности элементов блок-схемы, посредствам которых записан алгоритм:

- *begin*/*end* – специальные маркеры, интересные лишь тем, что алгоритм выполняется последовательно по некоторому «пути», который всегда выходит из *begin* и оканчивается в *end*;
- *var_in* – данный элемент означает, что некоторая переменная получает конкретную интерпретацию. Обозначим это действие как *mark_user_put*;
- *var_out* – элемент обозначает вывод (передачу «наружу») значения переменной, т.е. является результатом (одной из координат результата, в многомерном случае), а значит, говоря на языке концепции *Q*-детерминанта, увеличивает размерность безусловного *Q*-терма *w*;
- *condition* – единственный элемент блок-схемы, имеющий 2 выхода, прохождение по каждому из которых характеризуется выполнением (или не выполнением) некоторого условия. Подобное прохождение влечет наращивание безусловного логического *Q*-терма *u* (условия объединяются с помощью конъюнкции);
- *execute* – элемент, изменяющий значение некоторой переменной. Обозначим это действие как *replace* (Отметим, что действие *replace* не осуществляет замену переменных, на которые ранее оказала воздействие операция *mark_user_put*).

Теперь можно сформулировать **общую идею метода (*ир*)**:

Шаг 1. Будем осуществлять обратный проход схемы алгоритма (двигаясь из *end* до *begin*). Будем сохранять действия, которые осуществляются при проходе, запоминая последовательность элементов блок-схемы.

Выбор обратного прохода обеспечивает то, что:

- будут получены все из возможных выводов алгоритма (в виде *Q*-термов), так как будут пройдены все ветвления приводящие к окончанию алгоритма;
- сформированный *Q* -терм не будет содержать избыточных действий, ведь все *replace* и *mark_user_put* проводятся в обратном порядке и учитываются только в случае существования соответствующих переменных
- сформированные пары (*u*, *w*) будут соответствовать действительно возможным вариантам прохода алгоритма.

Шаг 2. Выполняем шаг 1, пока у элемента лишь одна входящая связь, если связей 2 – переходим к шагу 3, если входных связей нет – переходим к шагу 6.

Шаг 3. «Обработка двоений». Проанализируем каждую из входных связей на цикличность (*is_while_path*). *Цикличной* будем называть такую связь, которая не может добраться до *begin*'а, не проходя через элемент, в который она входит.

Шаг 4. «Разрезка схемы» (*cut*). Сначала осуществляем разрезку блок-схемы от *end*'а до текущего элемента. Получаем некую «*head*»-часть схемы. Далее разрезаем схему по каждой из входящих связей. В случае если связь нецикличная, она будет являться одним из линейных продолжений («*tail*»-часть). Если же связь является цикличной, то «распрямляем» ее, получая «*body*»-часть. При каждой порезке дополняем получаемые схемы фиктивными терминирующими элементами.

Шаг 5. «Комбинирование частей». Предположим, что на шаге 4 было сформировано: *k tail*-частей, *l body*-частей и одна *head*-часть, где $k>0$ (k не может быть 0, иначе, получилось бы, что данный элемент при прямом проходе не достижим – что недопустимо по построению блок-схемы). Тогда получим *k* последовательностей элементов блок-схемы, каждая из которых имеет следующую структуру: в качестве «начала» выступает *head*-часть, затем идет множество, состоящее из *l body*-частей и, окончательно, одной *i*-ой *tail*-части, где $i=1,\ldots,k$.

Отметим, что к *body* и *tail*-частям, полученным на шаге 4, был рекурсивно применен данный алгоритм, начиная с шага 1, таким образом, что каждая из *body* или *tail*-части выступала как полноценная блок-схема, что допустимо т.к. при осуществлении разрезки схемы, получается блок-схема, удовлетворяющая всем ограничениям, наложенными в 2.1.; исключение составляет ограничение на количество исходящих связей у *condition*: после *cut* могут получиться *condition*'ы, обладающие лишь одной исходящей связью; однако это не должно смущать – ведь именно это и гарантирует «выпрямление» схемы и избавление от «двоений».

Шаг 6. «Достигли *begin*». Закончить обработку текущей схемы. Сформировать результат.

После выполнения данного метода, получаем множество иерархических структур, используя которые можно построить *Q*-детерминант исследуемого алгоритма. Следующие две части посвящены описанию полученных объектов и методике построения с их помощью *Q* -термов.

2.3. Вспомогательная структура ActionList

Для хранения операций блок-схемы будем использовать абстракцию *ActionList*.

ActionList (AL) - структура, хранящая последовательность и очередность действий, правила их взаимоследования и их характер. Общая формула данной структуры выглядит так:

AL ::= [*H*] {*B*} [*T*], где *B* и *T* являются *AL*.

Все части AL являются факультативными, однако, предполагаем, что хотя бы 1 часть все-таки есть (иначе данный AL безрезультативен).

Опишем структуру AL подробнее:

H («*head*») – линейная часть AL. Представляет собой приращения *du* и *dw* *Q*-терма, а также *последовательность изменений*, проводимых операциями *replace* и *mark_user_put*. Данная часть может входить в текущий AL не более чем в 1-м экземпляре.

B («*body*») – цикличная часть AL. Может входить в текущий AL произвольное (конечное) число раз. Каждая из B частей сама является AL.

T («*tail*») – линейное продолжение AL, само являющееся AL. Также как и H, эта часть входит в текущий AL не более чем в 1-м экземпляре.

Данная структура позволяет реализовывать описанный в 2.2. *up*-метод, в виде формирования иерархической структуры вложенных AL. Рассмотрим шаги 5 и 6 упомянутого метода подробнее.

Так, в шаге 5, при комбинировании результатов разрезки схемы, выполняются следующие действия:

1. Создаем *k* (по числу *tail*-частей) различных AL;
2. В часть H помещаем *head*-часть схемы;
3. В часть B помещаем все *l* результатов применения *up*-метода к имеющимся *body*-частям;
4. В часть T помещаем результат применения *up*-метода к *i*-ой *tail*-части, где $i=1,\ldots,k$.

На шаге 6, при достижении *begin*'а, помещаем текущий результат в часть H.

Определившись со структурой AL, опишем особенности его применения при построении *Q*-термов.

2.4. Построение *Q*-детерминанта по набору ActionList'ов

Исходя из методики, на основании которой производилось построение *ActionList*'ов, а также характера хранимой в части H информации (приращения *du* и *dw* *Q*-терма и перечень замен) можно сформулировать идею построения *Q*-детерминанта.

Идея. Часть H хранит некоторые изменения, которые требуется применить к исходному (возможно «пустому») *Q*-терму. Результат этих изменений нужно подать на вход части B (которая будет рассматриваться как отдельные AL, содержащие внутри себя части H, со списком изменений). Далее получившийся результат подается на вход части T и уже он является конченым. Если какой-то из частей AL нет, то будем пропускать ее «выполнение». Отметим, что для частей B и T требуется выполнить перечисленные действия рекурсивно.

Таким образом, имея множество AL для некоторого алгоритма, можно построить *Q*-детерминант данного алгоритма.

Заключение

Данная работа посвящена определению ресурса параллелизма алгоритмов на базе концепции *Q*-детерминанта. В работе предлагается метод построения *Q*-детерминанта для произвольного алгоритма. В рамках данного метода предлагается обобщенная форма записи алгоритмов (в виде блок-схем со специальным алфавитом) и альтернативная форма записи алгоритма в виде ActionList'а. Приводятся методы преобразования алгоритма сначала к форме AL, а затем к форме *Q*-детерминанта. В дальнейшем, для данного алгоритма могут быть получены оценки эффективности, построена и проанализирована его максимально быстрая реализация [1].

На основе предлагаемой методики разработано приложение, автоматизирующее построение *Q*-детерминанта. Для того чтобы убедится в адекватности и эффективности предложенного метода, проведены контрольные эксперименты, демонстрирующие построение AL и *Q*-детерминантов произвольных алгоритмов. Результаты всех тестов совпали с ожидаемыми, что доказывает адекватность разработанной методики построения *Q*-детерминанта.

Предлагаемый метод может быть использован для широкого класса алгоритмов (не обязательно численных), для приведения их к виду *Q'*-детерминанта, где в качестве *Q'* может выступать произвольное множество операций. Так, можно расширить множество *Q*, введя в него в качестве базовых операций – операции взятия обратного (по умножению), операцию вычисления синуса (или совсем уж «нетривиальных» операций, типа вычисления корней полиномов *n*-ой степени – набор операций определяется возможностями целевого «вычислителя», на базе которого, будет реализовываться данный алгоритм). Таким образом, один и тот же алгоритм может иметь два принципиально разных *Q*-детерминанта (для *Q* и *Q'*), и обладать различными сложностными характеристиками. Подобным образом, можно выявить «оптимальное» *Q''*, позволяющее получать приемлемые значения данных характеристик, для некой целевой группы алгоритмов, распараллеливание которой наиболее критично на данной вычислительной системе.

Также, если для решения некоторой проблемы существует несколько возможных алгоритмов, предлагаемая методика позволяет выявить среди них наиболее подходящие с точки зрения возможности распараллеливания под архитектуру интересующей вычислительной системы.

ЛИТЕРАТУРА:

1. Алеева В.Н.Анализ параллельных численных алгоритмов : Препринт №590. Новосибирск, 1985. 23 с. В надзаг.: ВЦ СО АН СССР.
2. ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. М., 1991. 27 с.
3. Ершов А.П. Вычислимость в произвольных областях и базисах // ИСИ СО РАН. [Новосибирск, 2005]. URL: http://start.iis.nsk.su/archive/eaindex.asp?did=27068 (дата обращения: 19.03.2010).