электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.021

Алгоритм распределения вычислений при условии различной стоимости переходов между операторами

***Карбовский С. В.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»

***Лобачев А. А.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»

Научный руководитель: Шайхутдинов А. А., ассистент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»
sh.artur@gmail.com

Параллельные вычисления — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно).

При разработке параллельных алгоритмов возникают следующие задачи:

- Разделение вычислений на независимые части
- Выделение информационных зависимостей
- Масштабирование
- Разделение подзадач между процессорами
- Обеспечение доступа к разделяемым ресурсам

Важной задачей является задача распределения параллельного алгоритма по узлам вычислительной сети (ВС). Важность данной задачи очевидна - именно от нее зависит эффективность использования системы, а значит и эффективность использования затраченных на ее создание ресурсов.

В данной работе рассматривается алгоритм распределения алгоритма по узлам ВС с условием минимизации количества узлов.

Задача разбиения алгоритма на нити описывается следующим образом:

1. Алгоритм представляется в виде множества операторов данного алгоритма, некоторые подмножества которого могут выполняться параллельно.

2. Каждый оператор имеет определённую стоимость выполнения (например, связанную с временем выполнения оператора).
3. Существуют зависимости по данным и по управлению между операторами:
    - Зависимость по данным предполагает, что некоторый оператор (приёмник) обрабатывает данные, которые являются результатом выполнения другого оператора (источника);
    - Зависимость по управлению предполагает, что в результате выбора, совершённого в некотором операторе (источнике), может быть выполнен один из списка других операторов (приёмников).

      Примечание. Зависимость по управлению соответствует логике условного перехода (блоки if *и* switch *в С-подобных языках программирования)*
4. В случае, если пара операторов, между которыми существует связь, выполняются на разных процессорах, существует ненулевая стоимость перехода между операторами.
5. Для каждого оператора вводятся понятия свёртки и развёртки [2]:
    - Свёрткой данного оператора называется существование некоторого множества операторов, для которых данный оператор является приёмником;
    - Развёрткой данного оператора называется существование некоторого множества операторов, для которых данный оператор является источником.
6. Также вводятся понятия элементарной свёртки и развёртки, являющиеся, соответственно, свёрткой с единственным источником и развёрткой с единственным приёмником [2].
7. На свёртках и развёртках, не являющихся элементарными, реализуются функции, аналогичные логическим функциям «И» и «ИЛИ»[2]:
    - Функция «И» на свёртке – оператор выполняется, если выполнены все операторы из свёртки;
    - Функция «ИЛИ» на свёртке – оператор выполняется, если выполнен хотя бы один оператор из свёртки;
    - Функция «И» на развёртке – после выполнения оператора выполняются все операторы из развёртки;

      Примечание. Часто данная функция описывает распараллеленные итерации цикла.
    - Функция «ИЛИ» на развёртке – после выполнения оператора выполняется какой-то один оператор из развёртки.

Примечание. Видно, что связь по управлению описывается функцией «ИЛИ» на развёртке.

8. Несколько операторов можно объединить в нить, если в данном алгоритме они не должны выполняться параллельно. В таком случае стоимость перехода между операторами равна нулю.

Входные данные представлены в виде граф-схемы некоторого алгоритма (ГСА), в котором вершины являются операторами данного алгоритма, а дуги отражают зависимости по управлению и данным между ними.

ГСА представляется парой:

$\Gamma = (V, E)$, где:

- V – множество вершин ГСА;
- E – множество дуг ГСА.

Вершину ГСА описывает кортеж:

$V_i = \left(i, C_i, F_i^I, F_i^O\right), i = \overline{1, n}$, где:

- i – номер вершины;
- C_i – стоимость выполнения оператора;
- F_i^I – функция свёртки оператора;
- F_i^O – функция развёртки оператора.

Функции свёртки и развёртки задаются следующим образом:

$F_i^I, F_i^O \in \{\&, \oplus, E, \emptyset\}$, где:

- $\&$ – функция «И»;
- $\oplus$ – функция «ИЛИ»;
- E – элементарная свёртка или развёртка;
- $\emptyset$ – отсутствие свёртки или рзвёртки.

Дуга ГСА, соединяющая -ю и j-ю вершины, также является кортежем, вида:

$E_{i,j} = \left(V_i, V_j, C_{i,j}\right), V_i \in V, V_j \in V, i \neq j$, где:

- V_i – начальная вершина дуги;
- V_j – конечная вершина дуги;
- $C_{i,j}$ – стоимость перехода между вершинами V_i, V_j

При реализации алгоритма удобно задавать множество дуг ГСА в виде матрицы следования или матрицы смежности.

Пример графического представления ГСА в специальной нотации представлен на рис. 1.

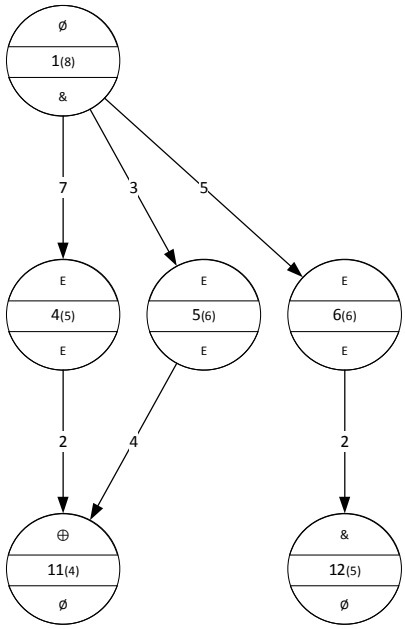

Рис. 1. Нотация для изображения ГСА

Результатом работы алгоритма является множество нитей, каждая из которых может быть выполнена на отдельном узле вычислительной системы. Каждая нить включает в себя множество операторов. Помимо множества нитей, в результат включается множество связей между нитями, то есть, другими словами, топология ВС.

Пример графического представления связей между нитями представлен на рис 2.

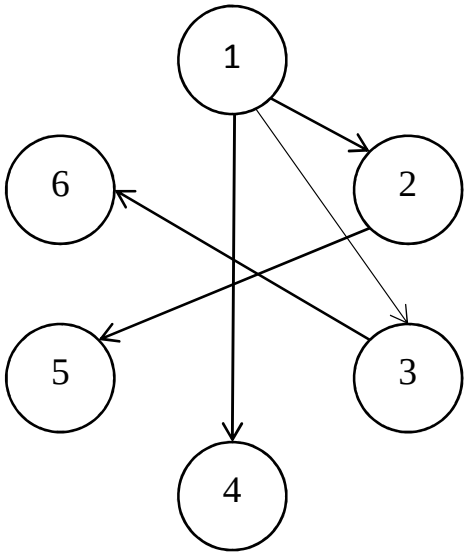

Рис. 2. Схема связей между нитями

В настоящее время при распределении операторов некоторого алгоритма по узлам ВС в основном применяют подход, основанный на анализе времени выполнения операторов алгоритма. Некоторые способы основаны на расчёте ранних времён выполнения операторов [3]; другие подходы ориентированы на уменьшении критического пути с рассмотрением каждого такта времени [1]. Существуют также адаптивные подходы, выполняющие планирование алгоритма в процессе выполнения алгоритма [4].

Стоит заметить, что вышеназванные алгоритмы не рассматривают в условиях задачи возможность различной стоимости перехода между операторами.

Разработанный алгоритм основан на комплексном стоимостном подходе. Данный подход расширяет временной анализ и предполагает анализировать как время выполнения операторов, так и стоимость перехода между ними. Алгоритм представлен в виде комплекса процедур, последовательно применяемых к ГСА:

- выделение нитей по логическим путям;
- выделение нитей по стоимости перехода;
- слияние нитей по времени;
- балансировка нитей.

Основной процедурой является выделение нитей по стоимости перехода. Данная процедура позволяет выделить первоначальный набор нитей, которые затем объединяются в соответствии с несколькими условиями.

Основное преимущество данного подхода – время работы распределяемого алгоритма по критическому пути не превышает идеального теоретического значения. Это связано с тем, что ни на одном шаге описываемого алгоритма не происходит сдвиг выполнения операторов во времени дальше критического времени.

Опишем процедуры, применяемые в алгоритме.

1. **Выделение нитей по логическим путям**

В случае, когда мы рассматриваем условный переход, возможно объединять в нить не пару операторов «предок-потомок», а сразу несколько операторов - предка и некоторое множество его потомков. Выделим критерий допустимости такого объединения на основе описанной выше модели.

Необходимым условием для объединения нескольких операторов-потомков является их независимость от всех операторов кроме их общего предка. Модель переводит это условие в плоскость, описываемую понятиями «свёртка» и «развёртка». Поскольку понятие «элементарная свёртка» описывает ситуацию, когда некоторый оператор зависим только от одного оператора, описанный выше критерий запишем следующим образом:

можно объединить оператор условного перехода с теми его потомками, свёртка которых является элементарной.

Вычислительная сложность процедуры - $O(n^2)$, где n - количество операторов.

2. **Выделение нитей по стоимости перехода**

На этом этапе работы алгоритм на каждом шаге избавляется от самой дорогой связи по данным. Работает это следующим образом: в начале связи по данным сортируются по убыванию, затем для каждой следующей связи пытаемся обнулить ее стоимость путем объединения вершин, которая она связывает, в одну нить. Объединение возможно в следующих случаях:

- Рассматриваем связь между двумя операторами, ни один из которых не принадлежит никакой нити - в этом случае создается новая нить;
- Рассматриваем связь между двумя операторами, один из которых завершает некоторую нить, а второй начинает некоторую другую нить - в этом случае две нити сливаются в одну;
- Рассматриваем связь между двумя операторами, один из которых является первым или последним оператором нити, а второй не принадлежит никакой нити - в этом случае оператор добавляется в нить.

Вычислительная сложность процедуры - $O(m^3)$, где m - количество связей.

3. **Слияние нитей по времени**

В результате применения предыдущей процедуры найдено множество нитей, для каждой из которых задано время начала и конца выполнения. Часто встречается ситуация, когда нити можно объединить, так как окончание первой раньше, чем начало работы второй, уменьшив при этом количество процессоров. Для этого используется алгоритм минимизации количества нитей [2]:

1. Сортируем нити по раннему сроку выполнения первого оператора нити.
2. Для каждой нити в отсортированном списке находим нить с минимальным временем начала, которое, однако, больше времени окончания текущей нити.
3. Объединяем текущую нить с найденной.

   Примечание. При объединении текущей и найденной нитей необходимо удалить найденную нить из списка нитей.

Вычислительная сложность алгоритма - $O(n^3 + m^3)$, где n - количество операторов, m – количество связей.

4. **Балансировка нитей**

Нередки случаи, когда в результате применения алгоритма слияния некоторое число нитей задействованы небольшое время относительно времени критического пути.

Очевидно, что остальное время узлы ВС, на которых выполняются эти нити, простаивают. В некоторых случаях возможно объединить такие нити с другими, уменьшив, тем самым, общее число нитей, не увеличивая при этом время критического пути. Объединение происходит, когда выполнение операторов нити-кандидата откладывается на время, необходимое для завершения нити, с которой нить-кандидата планируется объединить.

В целях упрощения алгоритма выборку нитей-кандидатов для объединения по следующему условию: нить завершается выходом из алгоритма.

Алгоритм определяется следующим образом:

1. Определяем нить, завершающий оператор которой завершает также критический путь.

   Примечание. Такой оператор имеет максимальное время завершения.
2. Сортируем остальные нити по времени завершения в порядке убывания.
3. Определяем список нитей-кандидатов и сортируем их по длительности выполнения.
4. Каждую нить из списка кандидатов объединяем с нитью из п.2 с максимальным временем завершения, при условии, что время завершения новой нити меньше времени завершения критического пути.

Вычислительная сложность алгоритма - $O(n^2 \log(n))$, где n - количество операторов.

В ходе выполнения курсовой работы по дисциплине «Вычислительные системы» была подготовлена реализация разработанного алгоритма на языке Python. Код программы доступен в репозитории GitHub: https://github.com/kser93/comp_sys

Вышеописанный алгоритм может применяться при разработке параллельных систем, например, для кластеров, систем реального времени.

Помимо распределения операторов по узлам ВС, алгоритм может применяться в широком спектре задач. Вот некоторые из них:

- Исследовательские задачи – моделирование сложных систем;
- Оптимизация бизнес-процессов – например, при моделировании конвейерной сборки. В данном случае можно провести аналогии «оператор – операция по обработки детали», «ребро – транспортировка детали». Соответственно, цена оператора будет трактоваться как цена обработки детали, а ребра – передачи детали.

Основное направление при дальнейшей модификации алгоритма заключается в добавлении адаптивности, то есть, динамического планирования. [4]

Список литературы

1. Andrews D.L., Thornton M.A. Graph Analysis and Transformation Techniques for Runtime Minimization in MultiThreaded Architectures // Thirtieth Annual Hawwaii International Conference on System Sciences: IEEE Computer Society, 1997.
2. Руденко Ю.М., Волкова Е.А. Вычислительные системы. М.: МГТУ им. Н.Э.Баумана, 2009. 179 с.
3. Руденко Ю.М. Планирование распределения программных модулей по процессорам вычислительной системы. Инженерный журнал: наука и инновации, 2013, вып. 11. Режим доступа: http://engjournal.ru/catalog/it/network/1010.html (дата обращения 05.05.2015).
4. Руденко Ю.М. Распределение программных модулей по узлам вычислительной сети с общим полем памяти для граф-схем параллельных алгоритмов // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2011. № 10. Режим доступа: http://www.technomag.edu.ru/doc/235842.html (дата обращения 16.04.2015).